Arria 10 Avalon-MM Interface for PCIe Solutions

User Guide

Last updated for Altera Complete Design Suite: 15.0
UG-01145_avmm
2015.06.05

Subscribe
Send Feedback

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Contents

Datasheet 1-1
Arria 10 Avalon-MM Interface for PCIe Datasheet 1-1
Features 1-2
Release Information 1-6
Device Family Support 1-7
Configurations 1-7
Example Designs 1-9
Debug Features 1-10
IP Core Verification 1-10
Compatibility Testing Environment 1-10
Performance and Resource Utilization 1-10
Recommended Speed Grades 1-11
Steps in Creating a Design for PCI Express 1-11

Getting Started with the Avalon-MM Arria 10 Hard IP for PCI Express 2-1
Running Qsys 2-2
Generating the Example Design 2-3
Understanding Simulation Log File Generation 2-5
Running a Gate-Level Simulation 2-5
Simulating the Single DWord Design 2-5
Generating Quartus II Synthesis Files 2-6
Creating a Quartus II Project 2-6
Adding Virtual Pin Assignment to the Quartus II Settings File (.qsf) 2-7
Compiling the Design 2-7
Programming a Device 2-7
Understanding Channel Placement Guidelines 2-7

Parameter Settings 3-1
Arria 10 Avalon-MM System Settings 3-1
Interface System Settings 3-4
Base Address Register (BAR) Settings 3-6
Device Identification Registers 3-7
PCI Express and PCI Capabilities Parameters 3-8
Device Capabilities 3-8
Error Reporting 3-9
Link Capabilities 3-10
MSI and MSI-X Capabilities 3-11
Slot Capabilities 3-12
Power Management 3-13

Physical Layout of Hard IP In Arria 10 Devices..... 4-1
Channel and Pin Placement for the Gen1, Gen2, and Gen3 Data Rates..... 4-5
Channel Placement and fPLL Usage for the Gen1 and Gen2 Data Rates..... 4-6
Channel Placement and fPLL and ATX PLL Usage for the Gen3 Data Rate..... 4-8

64- or 128-Bit Avalon-MM Interface to the Application Layer..... 5-1
32-Bit Non-Bursting Avalon-MM Control Register Access (CRA) Slave Signals..... 5-3
RX Avalon-MM Master Signals..... 5-3
64- or 128-Bit Bursting TX Avalon-MM Slave Signals..... 5-5
Clock Signals..... 5-8
Reset..... 5-8
Interrupts for Endpoints when Multiple MSI/MSI-X Support Is Enabled..... 5-10
Hard IP Reconfiguration Interface..... 5-11
Physical Layer Interface Signals..... 5-13
Serial Data Signals..... 5-13
PIPE Interface Signals..... 5-14
Test Signals..... 5-19

Registers..... 6-1
Correspondence between Configuration Space Registers and the PCIe Specification..... 6-1
Type 0 Configuration Space Registers..... 6-5
Type 1 Configuration Space Registers..... 6-6
PCI Express Capability Structures..... 6-6
Altera-Defined VSEC Registers..... 6-9
CvP Registers..... 6-10
64- or 128-Bit Avalon-MM Bridge Register Descriptions..... 6-13
Avalon-MM to PCI Express Interrupt Registers..... 6-15
Programming Model for Avalon-MM Root Port..... 6-26
Sending a Write TLP..... 6-27
Sending a Read TLP or Receiving a Non-Posted Completion TLP..... 6-28
PCI Express to Avalon-MM Interrupt Status and Enable Registers for Root Ports..... 6-28
Root Port TLP Data Registers..... 6-29
Uncorrectable Internal Error Mask Register..... 6-31
Uncorrectable Internal Error Status Register..... 6-32
Correctable Internal Error Mask Register..... 6-33
Correctable Internal Error Status Register..... 6-34

Arria 10 Reset and Clocks..... 7-1
Reset Sequence for Hard IP for PCI Express IP Core and Application Layer..... 7-2
Clocks..... 7-4
Clock Domains..... 7-4
Clock Summary..... 7-6

Interrupts for Endpoints..... 8-1

Enabling MSI or Legacy Interrupts 8-2
Generation of Avalon-MM Interrupts 8-3
Interrupts for Endpoints Using the Avalon-MM Interface with Multiple MSI/MSI-X Support 8-3

Error Handling 9-1
Physical Layer Errors 9-2
Data Link Layer Errors 9-2
Transaction Layer Errors 9-3
Error Reporting and Data Poisoning 9-6
Uncorrectable and Correctable Error Status Bits 9-7

IP Core Architecture 10-1
Top-Level Interfaces 10-3
Avalon-MM Interface 10-3
Clocks and Reset 10-3
Interrupts 10-3
PIPE 10-3
Data Link Layer 10-4
Physical Layer 10-6
32-Bit PCI Express Avalon-MM Bridge 10-8
Avalon-MM Bridge TLPs 10-11
Avalon-MM-to-PCI Express Write Requests 10-11
Avalon-MM-to-PCI Express Upstream Read Requests 10-11
PCI Express-to-Avalon-MM Read Completions 10-12
PCI Express-to-Avalon-MM Downstream Write Requests 10-12
PCI Express-to-Avalon-MM Downstream Read Requests 10-12
Avalon-MM-to-PCI Express Read Completions 10-13
PCI Express-to-Avalon-MM Address Translation for 32-Bit Bridge 10-13
Minimizing BAR Sizes and the PCIe Address Space 10-15
Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing 10-17
Completer Only Single Dword Endpoint 10-19
RX Block 10-20
Avalon-MM RX Master Block 10-20
TX Block 10-21
Interrupt Handler Block 10-21

Design Implementation 11-1
Making Pin Assignments to Assign I/O Standard to Serial Data Pins 11-1
Recommended Reset Sequence to Avoid Link Training Issues 11-1
SDC Timing Constraints 11-2

Throughput Optimization 12-1
Throughput of Posted Writes 12-3
Throughput of Non-Posted Reads 12-3

Optional Features..... 13-1
Configuration via Protocol (CvP) 13-1
ECRC 13-2
ECRC on the RX Path 13-2
ECRC on the TX Path 13-3

Avalon-MM Testbench and Design Example 14-1
Arria 10 Avalon-MM Endpoint Testbench 14-2
Arria 10 Avalon-MM Root Port Testbench 14-4
Endpoint Design Example..... 14-4
BAR/Address Map 14-6
Avalon-MM Test Driver Module 14-7
DMA Write Cycles 14-8
DMA Read Cycles 14-10
Avalon-MM Root Port Design Example 14-12
Root Port BFM 14-14
BFM Memory Map 14-16
Configuration Space Bus and Device Numbering 14-16
Configuration of Root Port and Endpoint 14-16
Issuing Read and Write Transactions to the Application Layer 14-21
BFM Procedures and Functions 14-22
ebfm_barwr Procedure 14-22
ebfm_barwr_imm Procedure 14-23
ebfm_barrd_wait Procedure 14-24
ebfm_barrd_nowt Procedure 14-25
ebfm_cfgwr_imm_wait Procedure 14-26
ebfm_cfgwr_imm_nowt Procedure 14-26
ebfm_cfgrd_wait Procedure 14-27
ebfm_cfgrd_nowt Procedure 14-28
BFM Configuration Procedures..... 14-29
BFM Shared Memory Access Procedures 14-31
BFM Log and Message Procedures 14-34
Verilog HDL Formatting Functions 14-38
Procedures and Functions Specific to the Chaining DMA Design Example..... 14-42
Setting Up Simulation..... 14-49
Changing Between Serial and PIPE Simulation 14-49
Using the PIPE Interface for Gen1 and Gen2 Variants 14-49
Viewing the Important PIPE Interface Signals..... 14-49
Disabling the Scrambler for Gen1 and Gen2 Simulations 14-49
Disabling 8B/10B Encoding and Decoding for Gen1 and Gen2 Simulations..... 14-50

Debugging 15-1
Simulation Fails To Progress Beyond Polling.Active State..... 15-1
Hardware Bring-Up Issues 15-1
Link Training 15-2

Use Third-Party PCIe Analyzer 15-2
BIOS Enumeration Issues 15-3

Frequently Asked Questions..... A-1

Lane Initialization and Reversal B-1

Additional Information.....C-1
Revision History for the Avalon-MM Interface.....C-1
How to Contact Altera..... C-4
Typographic Conventions..... C-5

Datasheet 1

2015.06.05

UG-01145_avmm Subscribe Send Feedback

Arria 10 Avalon-MM Interface for PCIe Datasheet

Altera® Arria® 10 FPGAs include a configurable, hardened protocol stack for PCI Express® that is compliant with *PCI Express Base Specification 3.0*.

The Hard IP for PCI Express IP core using the Avalon ® Memory-Mapped (Avalon-MM) interface removes some of the complexities associated with the PCIe protocol. For example, it handles all of the Transaction Layer Protocol (TLP) encoding and decoding. Consequently, you can complete your design more quickly. The Avalon-MM interface is implemented as a bridge in soft logic. It is available in Qsys.

Figure 1-1: Arria 10 PCIe Variant with Avalon-MM Interface

The following figure shows the high-level modules and connecting interfaces for this variant.

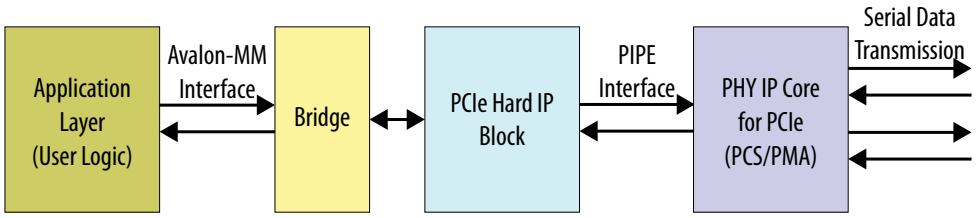


Table 1-1: PCI Express Data Throughput

The following table shows the aggregate bandwidth of a PCI Express link for Gen1, Gen2, and Gen3 for 1, 2, 4, and 8 lanes. The protocol specifies 2.5 giga-transfers per second for Gen1, 5.0 giga-transfers per second for Gen2, and 8.0 giga-transfers per second for Gen3. This table provides bandwidths for a single transmit (TX) or receive (RX) channel. The numbers double for duplex operation. Gen1 and Gen2 use 8B/10B encoding which introduces a 20% overhead. In contrast, Gen3 uses 128b/130b encoding which reduces the data throughput lost to encoding to less than 1%.

	Link Width in Gigabits Per Second (Gbps)			
	x1	x2	x4	x8
PCI Express Gen1 (2.5 Gbps)	2	4	8	16

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

	Link Width in Gigabits Per Second (Gbps)			
	x1	x2	x4	x8
PCI Express Gen2 (5.0 Gbps)	4	8	16	32
PCI Express Gen3 (8.0 Gbps)	7.87	15.75	31.51	63

Refer to *AN 456: PCI Express High Performance Reference Design* for more information about calculating bandwidth for the hard IP implementation of PCI Express in many Altera FPGAs, including the Arria 10 Hard IP for PCI Express IP core.

Related Information

- **PCI Express Base Specification 3.0**
- **AN 456: PCI Express High Performance Reference Design**
- **Creating a System with Qsys**

Features

New features in the Quartus® II 15.0 software release:

- Added **Enable Altera Debug Master Endpoint (ADME)** parameter to support optional Native PHY register programming with the Altera System Console.
- Dynamic generation of Qsys design examples using the parameters that you specify.
- Added Root Port support for transmitting messages of length greater than one dword.

The Arria 10 Hard IP for PCI Express with the Avalon-MM interface supports the following features:

- Complete protocol stack including the Transaction, Data Link, and Physical Layers implemented as hard IP.
- Support for ×1, ×2, ×4, and ×8 configurations with Gen1, Gen2, or Gen3 lane rates for Root Ports and Endpoints.
- Dedicated 16 KByte receive buffer.
- Optional support for Configuration via Protocol (CvP) using the PCIe link allowing the I/O and core bitstreams to be stored separately.
- Support for 32- or 64-bit addressing for the Avalon-MM interface to the Application Layer.
- Qsys example designs demonstrating parameterization, design modules, and connectivity.
- Extended credit allocation settings to better optimize the RX buffer space based on application type.
- Optional end-to-end cyclic redundancy code (ECRC) generation and checking and advanced error reporting (AER) for high reliability applications.
- Easy to use:
  - Flexible configuration.
  - No license requirement.
  - Example designs to get started.

Table 1-2: Feature Comparison for all Hard IP for PCI Express IP Cores

The table compares the features of the four Hard IP for PCI Express IP Cores.

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA	Avalon-ST Interface with SR-IOV
IP Core License	Free	Free	Free	Free
Native Endpoint	Supported	Supported	Supported	Supported
Legacy Endpoint (1)	Supported	Not Supported	Not Supported	Not Supported
Root port	Supported	Supported	Not Supported	Not Supported
Gen1	×1, ×2, ×4, ×8	×1, ×2, ×4, ×8	Not Supported	×8
Gen2	×1, ×2, ×4, ×8	×1, ×2, ×4, ×8	×4, ×8	×4, ×8
Gen3	×1, ×2, ×4, ×8	×1, ×2, ×4	×2, ×4, ×8	×2, ×4, ×8
64-bit Application Layer interface	Supported	Supported	Not supported	Not supported
128-bit Application Layer interface	Supported	Supported	Supported	Supported
256-bit Application Layer interface	Supported	Not Supported	Supported	Supported
Maximum payload size	128, 256, 512, 1024, 2048 bytes	128, 256 bytes	128, 256 bytes	128, 256 bytes
Number of tags supported for non-posted requests	256	8	16	256

(1) Not recommended for new designs.

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA	Avalon-ST Interface with SR-IOV
Automatically handle out-of-order completions (transparent to the Application Layer)	Not supported	Supported	Supported	Not supported
Automatically handle requests that cross 4 KByte address boundary (transparent to the Application Layer)	Not supported	Supported	Supported	Not Supported
Polarity Inversion of PIPE interface signals	Supported	Supported	Supported	Supported
Number of MSI requests	1, 2, 4, 8, 16, or 32	1, 2, 4, 8, 16, or 32	1, 2, 4, 8, 16, or 32	1, 2, 4, 8, 16, or 32 (for Physical Functions)
MSI-X	Supported	Supported	Supported	Supported
Legacy interrupts	Supported	Supported	Supported	Supported
Expansion ROM	Supported	Not supported	Not supported	Not supported

Table 1-3: TLP Support Comparison for all Hard IP for PCI Express IP Cores

The table compares the TLP types that the four Hard IP for PCI Express IP Cores can transmit. Each entry indicates whether this TLP type is supported (for transmit) by endpoints (EP), Root Ports (RP), or both (EP/RP).

Transaction Layer Packet type (TLP) (transmit support)	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA	Avalon-ST Interface with SR-IOV
Memory Read Request (`Mrd`)	EP/RP	EP/RP	EP	EP

Transaction Layer Packet type (TLP) (transmit support)	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA	Avalon-ST Interface with SR-IOV
Memory Read Lock Request (MRdLk)	EP/RP		EP	EP
Memory Write Request (MWr)	EP/RP	EP/RP	EP	EP
I/O Read Request (IORd)	EP/RP	EP/RP		EP
I/O Write Request (IOWr)	EP/RP	EP/RP		EP
Config Type 0 Read Request (CfgRd0)	RP	RP		EP
Config Type 0 Write Request (CfgWr0)	RP	RP		EP
Config Type 1 Read Request (CfgRd1)	RP	RP		EP
Config Type 1 Write Request (CfgWr1)	RP	RP		EP
Message Request (Msg)	EP/RP	EP/RP		EP
Message Request with Data (MsgD)	EP/RP	EP/RP		EP
Completion (Cpl)	EP/RP	EP/RP	EP	EP
Completion with Data (CplD)	EP/RP		EP	EP
Completion-Locked (CplLk)	EP/RP			EP
Completion Lock with Data (CplDLk)	EP/RP			EP

Transaction Layer Packet type (TLP) (transmit support)	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA	Avalon-ST Interface with SR-IOV
Fetch and Add AtomicOp Request (`FetchAdd`)	EP			

The *Arria 10 Avalon-MM Interface for PCIe Solutions User Guide* explains how to use this IP core and not the PCI Express protocol. Although there is inevitable overlap between these two purposes, use this document only in conjunction with an understanding of the *PCI Express Base Specification*.

Note: This release provides separate user guides for the different variants. The *Related Information* provides links to all versions.

Related Information

- **Arria 10 Avalon-ST Interface for PCIe Solutions User Guide**
- **Arria 10 Avalon-ST Interface with SR-IOV for PCIe Solutions User Guide**
- **Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide**

Release Information

Table 1-4: Hard IP for PCI Express Release Information

Item	Description
Version	15.0
Release Date	May 2015
Ordering Codes	No ordering code is required
Product IDs	There are no encrypted files for the Arria 10 Hard IP for PCI Express. The Product ID and Vendor ID are not required because this IP core does not require a license.
Vendor ID	

Device Family Support

Table 1-5: Device Family Support

Device Family	Support
Arria 10	Preliminary. The IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.
Other device families	Refer to the *Altera's PCI Express IP Solutions* web page for support information on other device families.

Related Information

- **Altera's PCI Express IP Solutions web page**

Configurations

The Avalon-MM Arria 10 Hard IP for PCI Express includes a full hard IP implementation of the PCI Express stack comprising the following layers:

- Physical (PHY), including:
  - Physical Media Attachment (PMA)
  - Physical Coding Sublayer (PCS)
- Media Access Control (MAC)
- Data Link Layer (DL)
- Transaction Layer (TL)

When configured as an Endpoint, the Arria 10 Hard IP for PCI Express using the Avalon-MM supports memory read and write requests and completions with or without data.

Figure 1-2: PCI Express Application with a Single Root Port and Endpoint

The following figure shows a PCI Express link between two Arria 10 FPGAs.

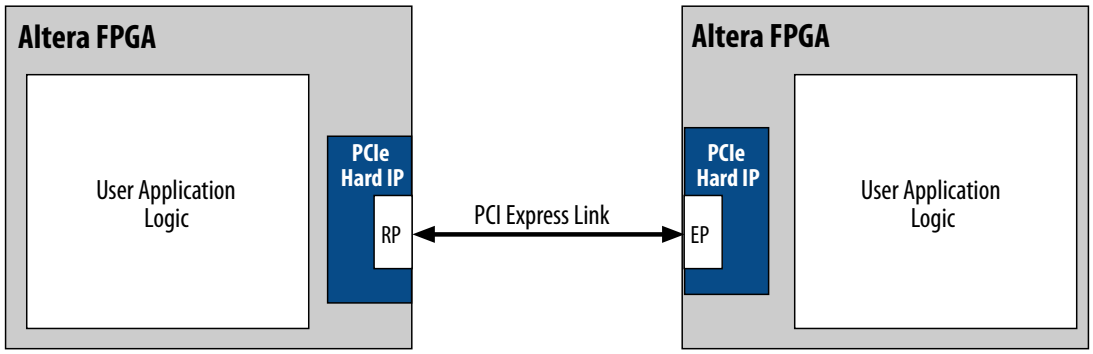
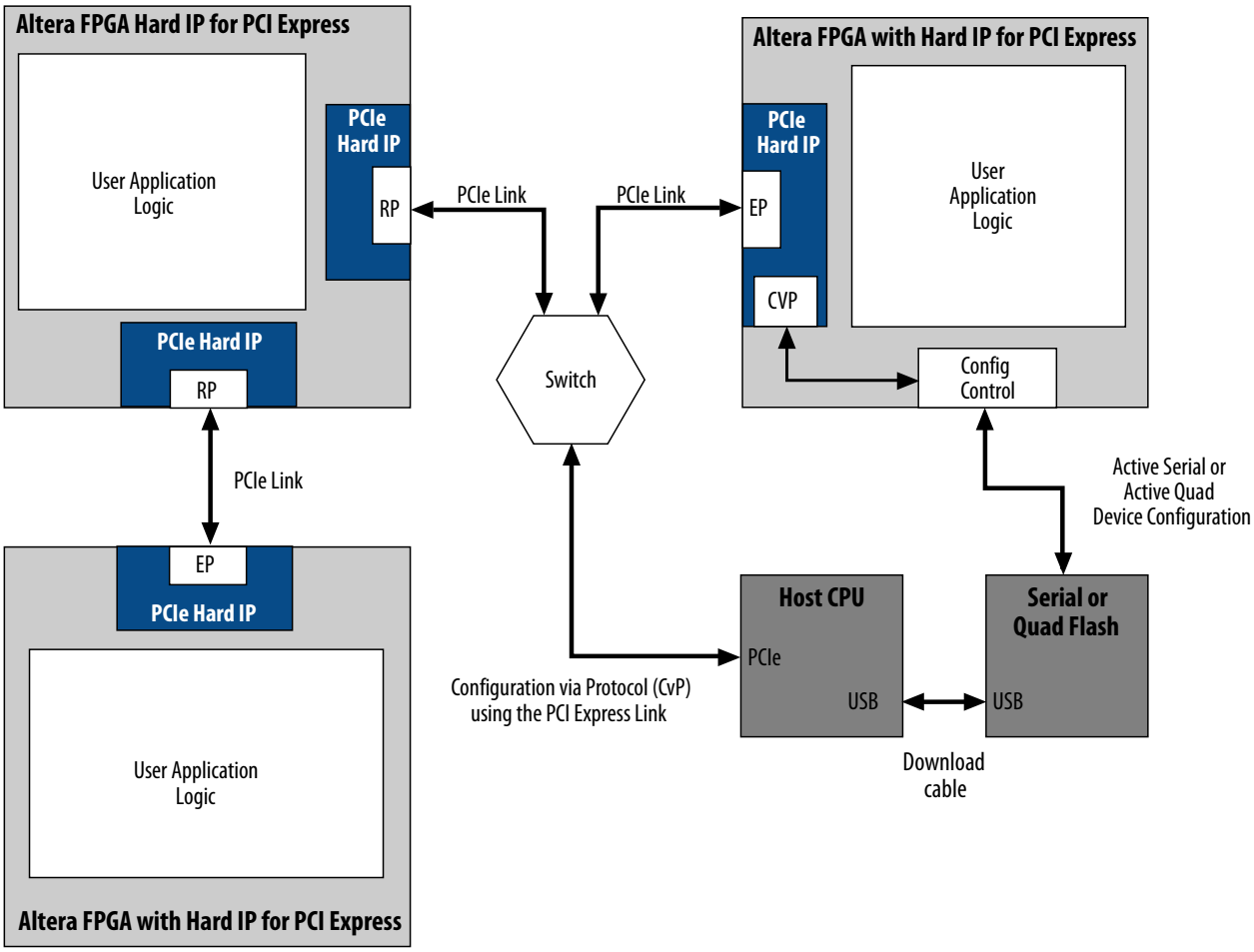


Figure 1-3: PCI Express Application Using Configuration via Protocol

The Arria 10 design below includes the following components:

- A Root Port that connects directly to a second FPGA that includes an Endpoint.
- Two Endpoints that connect to a PCIe switch.
- A host CPU that implements CvP using the PCI Express link connects through the switch. For more information about configuration over a PCI Express link, refer to **Configuration via Protocol (CvP)** on page 13-1.



Related Information

Configuration via Protocol (CvP)Implementation in Altera FPGAs User Guide

Example Designs

Qsys example designs are available for the Avalon-MM Arria 10 Hard IP for PCI Express IP Core. You can download them from the ***<install_dir>*/ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10** directory:

When you click the **Example Design** button in the Parameter Editor, you are prompted to specify the example design location. After example design generation completes, this directory contains one or two example designs. One is the example design from the ***<install_dir>*** that best matches the current parameter settings. This example design provides a static DUT. The other example design is a customized example design that matches your parameter settings exactly; starting in the Quartus II software v15.0, this feature is available for most but not all IP core variations. If this feature is not available for your particular parameter settings, the Parameter Editor displays a warning.

Related Information

Getting Started with the Avalon-MM Arria 10 Hard IP for PCI Express on page 2-1

Debug Features

Debug features allow observation and control of the Hard IP for faster debugging of system-level problems.

Related Information

Debugging on page 15-1

IP Core Verification

To ensure compliance with the PCI Express specification, Altera performs extensive verification. The simulation environment uses multiple testbenches that consist of industry-standard bus functional models (BFMs) driving the PCI Express link interface. Altera performs the following tests in the simulation environment:

- Directed and pseudorandom stimuli are applied to test the Application Layer interface, Configuration Space, and all types and sizes of TLPs
- Error injection tests that inject errors in the link, TLPs, and Data Link Layer Packets (DLLPs), and check for the proper responses
- PCI-SIG® Compliance Checklist tests that specifically test the items in the checklist
- Random tests that test a wide range of traffic patterns

Altera provides example designs that you can leverage to test your PCBs and complete compliance base board testing (CBB testing) at PCI-SIG, upon request.

Compatibility Testing Environment

Altera has performed significant hardware testing to ensure a reliable solution. In addition, Altera internally tests every release with motherboards and PCI Express switches from a variety of manufacturers. All PCI-SIG compliance tests are run with each IP core release.

Performance and Resource Utilization

Because the PCIe protocol stack is implemented in hardened logic, it uses no core device resources (no ALMs and no embedded memory).

The Avalon-MM soft logic bridge functions as a front end to the hardened protocol stack. The following table shows the typical device resource utilization for selected configurations using the current version of the Quartus II software. With the exception of M20K memory blocks, the numbers of ALMs and logic registers are rounded up to the nearest 50.

Table 1-6: Performance and Resource Utilization Avalon-MM Hard IP for PCI Express

Interface Width	ALMs	M20K Memory Blocks	Logic Registers
Avalon-MM Bridge			
64	1100	17	1500

Interface Width	ALMs	M20K Memory Blocks	Logic Registers
128	1900	25	2900
Avalon-MM Interface–Completer Only			
64	650	8	1000
128	1400	12	2400
Avalon-MM–Completer Only Single Dword			
64	250	0	350

Related Information

Fitter Resources Reports

Recommended Speed Grades

Recommended speed grades are pending characterization of production Arria 10 devices.

Related Information

- Area and Timing Optimization
- Altera Software Installation and Licensing Manual
- Setting up and Running Analysis and Synthesis

Steps in Creating a Design for PCI Express

Before you begin

Select the PCIe variant that best meets your design requirements.

- Is your design an Endpoint or Root Port?
- What Generation do you intend to implement?
- What link width do you intend to implement?
- What bandwidth does your application require?
- Does your design require CvP?

1. Select parameters for that variant.
2. Simulate using an Altera-provided example design. All of Altera's static PCI Express example designs are available under **<*install_dir*>/ip/altera/altera_pcie/**. Alternatively, generate an example design that matches your parameter settings, or create a simulation model and use your own custom or third-party BFM. The Qsys Generate menu generates simulation models. Altera supports ModelSim-Altera

for all IP. The PCIe cores support the Aldec RivieraPro, Cadence NCsim, Mentor Graphics ModelSim, and Synopsys VCS and VCS-MX simulators.

3. Compile your design using the Quartus II software. If the versions of your design and the Quartus II software you are running do not match, regenerate your PCIe design.
4. Download your design to an Altera development board or your own PCB. Click on the *All Development Kits* link below for a list of Altera's development boards.
5. Test the hardware. You can use Altera's SignalTap® II Logic Analyzer or a third-party protocol analyzer to observe behavior.
6. Substitute your Application Layer logic for the Application Layer logic in Altera's testbench. Then repeat Steps 3–6. In Altera's testbenches, the PCIe core is typically called the DUT (device under test). The Application Layer logic is typically called APPS.

Related Information

- **Parameter Settings** on page 3-1
- **Getting Started with the Avalon-MM Arria 10 Hard IP for PCI Express** on page 2-1
- **All Development Kits**

Getting Started with the Avalon-MM Arria 10 Hard IP for PCI Express

2

2015.06.05

UG-01145_avmm

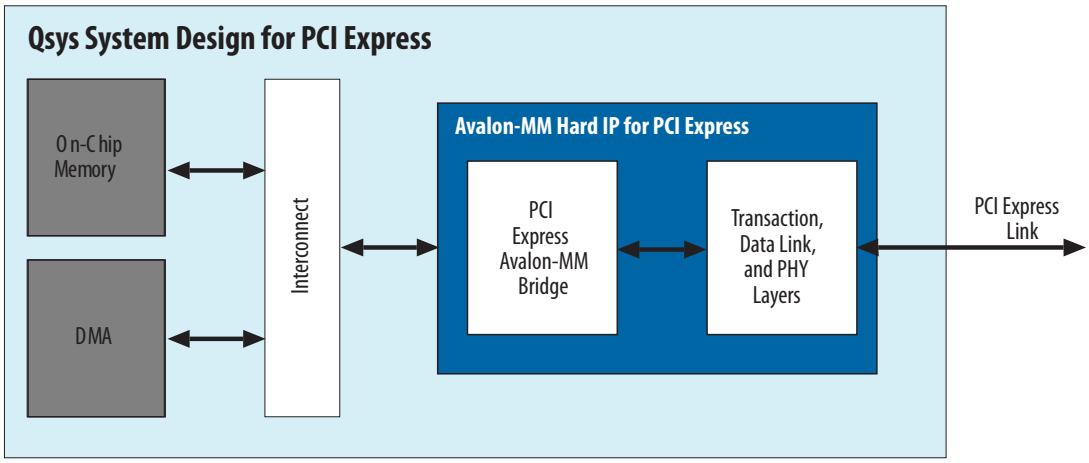
Subscribe Send Feedback

This Qsys design example provides detailed step-by-step instructions to generate a Qsys system. When you install the Quartus II software you also install the IP Library. This installation includes design examples for the Avalon-MM Arria 10 Hard IP for PCI Express in the **<*install_dir*>/ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10** directory. This walkthrough uses a Gen2 x4 Endpoint, **ep_g2x4_avmm128.qsys**.

The design examples contain the following components:

- Avalon-MM Arria 10 Hard IP for PCI Express IP core
- On-Chip memory
- DMA controller

Figure 2-1: Qsys Generated Endpoint



The design example transfers data between an on-chip memory buffer located on the Avalon-MM side and a PCI Express memory buffer located on the root complex side. The data transfer uses the DMA component which is programmed by the PCI Express software application running on the Root Complex processor.

Related Information

- **Generating the Example Design** on page 2-3

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

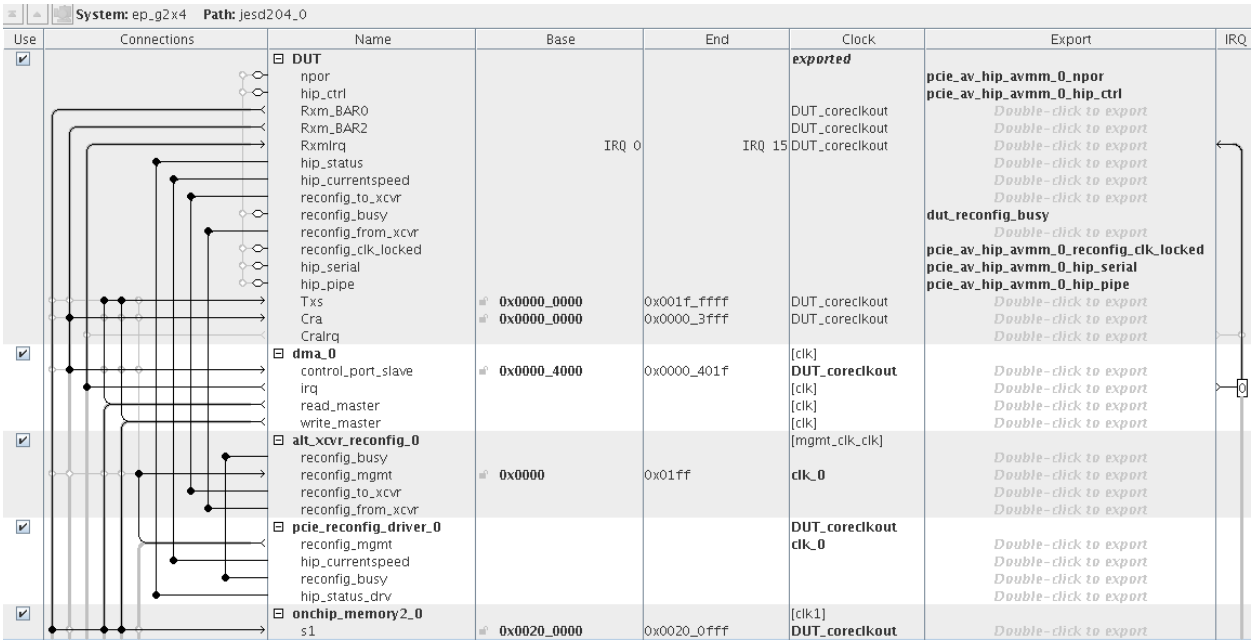
- **Creating a System with Qsys**
  This document provides an introduction to Qsys.

Running Qsys

1. Choose **Programs > Altera > Quartus II><version_number>** (Windows Start menu) to run the Quartus II software. Alternatively, you can also use the Quartus II Web Edition software.
2. On the File menu, select **New**, then **Qsys System File**.
3. Open the **ep_g2x4_avmm128.qsys** example design.

The following figure shows a Qsys system that includes the Transceiver Reconfiguration Controller and the Altera PCIe Reconfig Driver IP Cores. The Transceiver Reconfiguration Controller performs dynamic reconfiguration of the analog transceiver settings to optimize signal quality. You must include these components to the Qsys system to run successfully in hardware.

Figure 2-2: Qsys Avalon-MM Design for PCIe with Transceiver Reconfiguration Components



Refer to *Creating a System with Qsys* in volume 1 of the *Quartus II Handbook* for more information about how to use Qsys. For an explanation of each Qsys menu item, refer to *About Qsys* in Quartus II Help.

Related Information

- **Creating a System with Qsys**
- **About Qsys**

Generating the Example Design

1. On the Generate menu, select **Generate Testbench System**. The **Generation** dialog box appears.
2. Under **Testbench System**, set the following options:
   a. For **Create testbench Qsys system**, select **Standard, BFMs for standard Qsys interfaces**.
   b. For **Create testbench simulation model**, select **Verilog**.
3. You can retain the default values for all other parameters.
4. Click **Generate**.
5. After Qsys reports **Generation Completed**, click **Close**.
6. On the File menu, click **Save**.

The following table lists the testbench and simulation directories Qsys generates.

Table 2-1: Qsys System Generated Directories

Directory	Location
Qsys system	**<*project_dir*>/ep_g2x4_avmm128_tb**
Simulation Directory	**<*project_dir*>/ep_g2x4_avmm128_tb/ep_g2x4_tb/sim/<*cad_vendor*>**

The design example simulation includes the following components and software:

- The Qsys system
- A testbench. You can view this testbench in Qsys by opening **<*project_dir*>/ep_g2x4_avmm128_tb/ep_g2x4_avmm128_tb.qsys**.
- The ModelSim software

Note: You can also use any other supported third-party simulator to simulate your design.

Complete the following steps to run the Qsys testbench:

1. In a terminal window, change to the **<*project_dir*>/ep_g2x4_avmm128_tb/ep_g2x4_avmm128_tb/sim/mentor** directory.
2. Start the ModelSim® simulator.
3. Type the following commands in a terminal window:
   a. `do msim_setup.tcl`
   b. `ld_debug`
   c. `run 140000 ns`

The driver performs the following transactions with status of the transactions displayed in the ModelSim simulation message window:

1. Various configuration accesses to the Avalon-MM Arria 10 Hard IP for PCI Express in your system after the link is initialized
2. Setup of the Address Translation Table for requests that are coming from the DMA component
3. Setup of the DMA controller to read 512 Bytes of data from the Transaction Layer Direct BFM shared memory
4. Setup of the DMA controller to write the same data back to the Transaction Layer Direct BFM shared memory
5. Data comparison and report of any mismatch

The following example shows the transcript from a successful simulation run.

Example 2-1: Transcript from ModelSim Simulation of Gen2 x4 Endpoint

```
# INFO:  3657 ns  RP LTSSM State: DETECT.ACTIVE
# INFO:  4425 ns  RP LTSSM State: POLLING.ACTIVE
# INFO: 17257 ns  RP LTSSM State: DETECT.QUIET
# INFO: 17353 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 17405 ns  RP LTSSM State: DETECT.QUIET
# INFO: 17485 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 18249 ns  RP LTSSM State: POLLING.ACTIVE
# INFO: 23685 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 28510 ns  RP LTSSM State: DETECT.QUIET
. . .
# INFO: 44777 ns  RP LTSSM State: POLLING.CONFIG
# INFO: 45865 ns  RP LTSSM State: CONFIG.LINKWIDTH.START
# INFO: 46213 ns  EP LTSSM State: CONFIG.LINKWIDTH.START
# INFO: 46885 ns  EP LTSSM State: CONFIG.LINKWIDTH.ACCEPT
# INFO: 47353 ns  RP LTSSM State: CONFIG.LINKWIDTH.ACCEPT
# INFO: 48549 ns  RP LTSSM State: CONFIG.LANENUM.WAIT
# INFO: 48825 ns  RP LTSSM State: CONFIG.LANENUM.ACCEPT
# INFO: 48869 ns  EP LTSSM State: CONFIG.LANENUM.ACCEPT
# INFO: 49145 ns  RP LTSSM State: CONFIG.LANENUM.WAIT
# INFO: 49337 ns  RP LTSSM State: CONFIG.LANENUM.ACCEPT
# INFO: 49657 ns  RP LTSSM State: CONFIG.COMPLETE
# INFO: 50149 ns  EP LTSSM State: CONFIG.COMPLETE
# INFO: 51429 ns  RP LTSSM State: CONFIG.IDLE
# INFO: 51456 ns  EP LTSSM State: CONFIG.IDLE
# INFO: 51609 ns  RP LTSSM State: L0
# INFO: 51909 ns  EP LTSSM State: L0
 . . .
# INFO: 82248 ns Completed configuration of Endpoint BARs.
# INFO: 83016 ns Starting Target Write/Read Test.
# INFO: 83016 ns Target BAR = 0
# INFO: 83016 ns Length = 000512,Start Offset=000000
# INFO: 85264 ns Target Write and Read compared okay
# INFO: 85264 ns Starting DMA Read/Write Test.
# INFO: 85264 ns Setup BAR = 2
# INFO: 85264 ns Length = 000512, Start Offset = 000000
# INFO: 88616 ns Interrupt Monitor: Interrupt INTA Asserted
# INFO: 88616 ns Clear Interrupt INTA
# INFO: 89400 ns Interrupt Monitor: Interrupt INTA Deasserted
# INFO: 92892 ns MSI received!
# INFO: 92896 ns   DMA Read and Write compared okay!
# SUCCESS: Simulation stopped due to successful completion!
# Break in Function ebfm_log_stop_sim at ./..//ep_g1x4_avmm64_tb/simulation/
submodules//altpcietb_bfm_log.v line 78
```

Related Information
Simulating Altera Designs

Understanding Simulation Log File Generation

Starting with the Quartus II 14.0 software release, simulation automatically creates a log file, **altpcie_monitor_<*dev*>_dlhip_tlp_file_log.log** in your simulation directory.

Table 2-2: Sample Simulation Log File Entries

Time	TLP Type	Payload (Bytes)	TLP Header
17989 RX	CfgRd0	0004	04000001_0000000F_01080008
17989 RX	MRd	0000	00000000_00000000_01080000
18021 RX	CfgRd0	0004	04000001_0000010F_0108002C
18053 RX	CfgRd0	0004	04000001_0000030F_0108003C
18085 RX	MRd	0000	00000000_00000000_0108000C

Running a Gate-Level Simulation

The PCI Express testbenches run simulations at the register transfer level (RTL). However, it is possible to create you own gate-level simulations. Contact your Altera Sales Representative for instructions and an example that illustrate how to create a gate-level simulation from the RTL testbench.

Simulating the Single DWord Design

You can use the same testbench to simulate the Completer-Only Single Dword IP core by changing the settings in the driver file.

1. In a terminal window, change to the **<*variant*>_tb/<*variant*>_tb/altera_pcie_a10_tbed_140/sim/** directory.
2. Open **altpcietb_bfm_driver_avmm.v** in your text editor.
3. To enable target memory tests and specify the completer-only single dword variant, specify the following parameters:
   a. `parameter RUN_TGT_MEM_TST = 1;`
   b. `parameter RUN_DMA_MEM_TST = 0;`
   c. `parameter AVALON_MM_LITE = 1;`
4. Change to the **<*variant*>_tb/<*variant*>_tb/sim/mentor** directory.
5. Start the ModelSim simulator.
6. To run the simulation, type the following commands in a terminal window:

a. `do msim_setup.tcl`
b. `ld_debug` (The debug suffix stops optimizations, improving visibility in the ModelSim waveforms.)
c. `run 140000 ns`

Generating Quartus II Synthesis Files

1. On the **Generate** menu, select **Generate HDL**.
2. For **Create HDL design files for synthesis**, select **Verilog**.
   You can leave the default settings for all other items.
3. Click **Generate** to generate files for Quartus II synthesis.
4. Click **Finish** when the generation completes.

Creating a Quartus II Project

You can create a new Quartus II project with the New Project Wizard, which helps you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity.

1. On the Quartus II File menu, click then **New Project Wizard**, then **Next**.
2. Click **Next** in the **New Project Wizard: Introduction** (The introduction does not appear if you previously turned it off.)
3. On the **Directory, Name, Top-Level Entity** page, enter the following information:
   a. For **What is the working directory for this project**, browse to **<*project_dir*>/ep_g2x4_avmm128/synth**.
   b. For **What is the name of this project**, select **ep_g2x4_avmm128.v** from the **<*project_dir*>/ep_g2x4_avmm128/synth** directory.
   c. For **Project Type** select **Empty project**.
4. Click **Next**.
5. On the **Add Files** page, add **<*project_dir*>/ep_g2x4_128avmm/ep_g2x4_avmm128.qip** to your Quartus II project.
6. Click **Next** to display the **Family & Device Settings** page.
7. On the **Device** page, choose the following target device family and options:
   a. In the **Family** list, select **Arria 10**.
   b. In the **Devices** list, select **All**.
   c. In the **Available devices** list, select the appropriate device. For Arria 10 ES2 development kits, select **10AX115S1F45I3SGE2**.
8. Click **Next** to close this page and display the **EDA Tool Settings** page.
9. From the **Simulation** list, select **ModelSim**. From the **Format** list, select the HDL language you intend to use for simulation.
10. Click **Next** to display the **Summary** page.
11. Check the **Summary** page to ensure that you have entered all the information correctly.

Adding Virtual Pin Assignment to the Quartus II Settings File (.qsf)

To compile successfully you must add a virtual pin assignment statement for the PIPE interface to your **.qsf** file. The PIPE interface is useful for debugging, but is not a top-level interface of the IP core.

1. Browse to the synthesis directory that includes the **.qsf** for your project, **<*project_dir*>/ep_g2x4_avmm128/synth**
2. Open **ep_g2x4_avmm128.qsf**.
3. Add the following assignment statement:
   ```
   set_instance_assignment -name VIRTUAL_PIN ON -to pcie_a10_hip_0_hip_pipe_*
   ```
4. Save the **.qsf** file.

Compiling the Design

1. Before compiling, you need to make a few changes to your top-level Verilog HDL file to create a design that you can successfully download to a PCB.
   a. In the **<*project_dir*>/ep_g2x4_avmm128/synth/**, open **ep_g2x4_avmm128.v.**
   b. Comment out the declaration for `pcie_a10_hip_0_hip_ctrl_test_in`.
   c. Add a wire `[31:0] pcie_a10_hip_0_hip_ctrl_test_in` declaration to the same the same file.
   d. Assign `pcie_a10_hip_0_hip_ctrl_test_in` = 0x000000A8.
   e. Connect `pcie_a10_hip_0_hip_ctrl_test_in` to the `test_in` port on the Arria 10 Hard IP for PCI Express instance.
2. On the Quartus II Processing menu, click **Start Compilation**.
3. After compilation, expand the **TimeQuest Timing Analyzer** folder in the Compilation Report. Note whether the timing constraints are achieved in the Compilation Report.

If your design does not initially meet the timing constraints, you can find the optimal Fitter settings for your design by using the Design Space Explorer. To use the Design Space Explorer, click **Launch Design Space Explorer** on the Tools menu.

Programming a Device

After you compile your design, you can program your targeted Altera device and verify your design in hardware.

For more information about programming Altera FPGAs, refer to *Quartus II Programmer*.

Related Information

Quartus II Programmer

Understanding Channel Placement Guidelines

Arria 10 transceivers are organized in banks of six channels. The transceiver bank boundaries are important for clocking resources, bonding channels, and fitting. Refer to the *Channel Placement for the Gen1 and Gen2 Data Rates* and *Channel Placment and fPLL and ATX PLL Usage for the Gen3 Data Rates* for illustrations of channel placement for x1, x2, x4, and x8 variants.

Related Information

- **Channel Placement and fPLL Usage for the Gen1 and Gen2 Data Rates** on page 4-6
- **Channel Placement and fPLL and ATX PLL Usage for the Gen3 Data Rate** on page 4-8

Parameter Settings 3

2015.06.05

UG-01145_avmm

Arria 10 Avalon-MM System Settings

Table 3-1: System Settings for PCI Express

Parameter	Value	Description
Number of Lanes	**x1, x2, x4, x8**	Specifies the maximum number of lanes supported.
Lane Rate	**Gen1 (2.5 Gbps)** **Gen2 (2.5/5.0 Gbps)** **Gen3 (2.5/5.0/8.0 Gbps)**	Specifies the maximum data rate at which the link can operate.
Port type	**Native Endpoint** **Root Port** **Legacy endpoint**	Specifies the port type. Altera recommends **Native Endpoint** for all new Endpoint designs. Select **Legacy Endpoint** only when you require I/O transaction support for compatibility. The Endpoint stores parameters in the Type 0 Configuration Space. The Root Port stores parameters in the Type 1 Configuration Space.
Application Interface Type	**Avalon-ST** **Avalon-MM** **Avalon-MM with DMA** **Avalon-ST with SR-IOV**	Selects either the Avalon-ST interface, Avalon-MM interface, Avalon-MM with DMA interface, or Avalon-ST with SR-IOV interface.
RX Buffer credit allocation - performance for received requests	**Minimum** **Low** **Balanced**	Determines the allocation of posted header credits, posted data credits, non-posted header credits, completion header credits, and completion data credits in the 16 KByte RX buffer. The 5 settings allow you to adjust the credit allocation to

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Parameter	Value	Description
	High **Maximum100 MHz**	optimize your system. The credit allocation for the selected setting displays in the message pane. Refer to the *Throughput Optimization* chapter for more information about optimizing performance. The *Flow Control* chapter explains how the **RX credit allocation** and the **Maximum payload RX Buffer credit allocation** and the **Maximum payload size** that you choose affect the allocation of flow control credits. You can set the **Maximum payload size** parameter on the **Device** tab. The **Message** window dynamically updates the number of credits for Posted, Non-Posted Headers and Data, and Completion Headers and Data as you change this selection. • **Minimum**—configures the minimum PCIe specification allowed for non-posted and posted request credits, leaving most of the RX Buffer space for received completion header and data. Select this option for variations where application logic generates many read requests and only infrequently receives single requests from the PCIe link. • **Low**—configures a slightly larger amount of RX Buffer space for non-posted and posted request credits, but still dedicates most of the space for received completion header and data. Select this option for variations where application logic generates many read requests and infrequently receives small bursts of requests from the PCIe link. This option is recommended for typical endpoint applications where most of the PCIe traffic is generated by a DMA engine that is located in the endpoint application layer logic. • **Balanced**—configures approximately half the RX Buffer space to received requests and the other half of the RX Buffer space to received completions. Select this option for variations where the received requests and received completions are roughly equal.
Use 62.5 MHz application clock	**On/Off**	This mode is only available only for Gen1 ×1.
Enable byte parity ports on Avalon-ST interface	**On/Off**	When **On**, the RX and TX datapaths are parity protected. Parity is odd. This parameter is only available for the Avalon-ST Arria 10 Hard IP for PCI Express.

Parameter	Value	Description
Enable multiple packets per cycle for the 256-bit interface	**On/Off**	When **On**, the 256-bit Avalon-ST interface supports the transmission of TLPs starting at any 128-bit address boundary, allowing support for multiple packets in a single cycle. To support multiple packets per cycle, the Avalon-ST interface includes 2 start of packet and end of packet signals for the 256-bit Avalon-ST interfaces. This feature is only supported for Gen3 x8.
Enable configuration via PCI Express (CvP)	**On/Off**	When **On**, the Quartus II software places the Endpoint in the location required for configuration via protocol (CvP). For more information about CvP, click the *Configuration via Protocol (CvP)* link below. A single hard IP block in each device includes the CvP functionality. Refer to the*Physical Layout of Hard IP in Arria 10 Devices* for more information.
Enable credit consumed selection port	**On/Off**	When you turn on this option, the core includes the `tx_cons_cred_sel` port. This parameter does not apply to the Avalon-MM interface.
Enable dynamic reconfiguration of PCIE read-only registers	**On/Off**	When **On**, you can use the Hard IP reconfiguration bus to dynamically reconfigure Hard IP read-only registers. For more information refer to *Hard IP Reconfiguration Interface*.
Enable Altera Debug Master Endpoint (ADME)	**On/Off**	When **On**, you can use the Altera System Console to read and write the embedded Arria 10 Native PHY registers.

Related Information

- **Physical Layout of Hard IP In Arria 10 Devices** on page 4-1
- **PCI Express Base Specification 3.0**
- **Arria 10 Transceiver PHY User Guide**
  Provides information about the ADME feature for Arria 10 devices.

Interface System Settings

Table 3-2: Interface System Settings

Parameter	Value	Description
Application Interface width	**64-bit** **128-bit** **256-bit**	Specifies the data width for the Application Layer to Transaction Layer interface. Refer to *Application Layer Clock Frequency for All Combinations of Link Width, Data Rate and Application Layer Interface Widths* for all legal combinations of data width, number of lanes, Application Layer clock frequency, and data rate. The **Avalon-MM with DMA** interface does not support the 64-bit interface width.
Avalon-MM address width	**32, 64**	Specifies the address width for Avalon-MM RX master ports that access Avalon-MM slaves in the Avalon address domain. When you select 32-bit addresses, the PCI Express Avalon-MM bridge performs address translation. When you specify 64-bits addresses, no address translation is performed in either direction. The destination address specified is forwarded to the Avalon-MM interface without any changes. For the Avalon-MM interface with DMA, this value must be set to **64**.
Enable completer-only Endpoint	**On/Off**	In this mode, the Hard IP can receive requests, but cannot initiate upstream requests. However, it can transmit completion packets on the PCI Express TX link. This mode removes the Avalon-MM TX slave port and thereby reduces logic utilization.
Enable completer-only Endpoint with 4-byte payload	**On/Off**	This is a non-pipelined version of **Completer Only** mode. At any time, only a single request can be outstanding. **Single dword completer** uses fewer resources than **completer only Endpoint**. This variant is targeted for systems that require simple read and write register accesses from a host CPU. If you select this option, the width of the data for RXM BAR masters is always 32 bits, regardless of the **Avalon-MM** width. For the Avalon-MM interface with DMA, this value must be **Off** .

Parameter	Value	Description
Enable control register access (CRA) Avalon-MM slave port	**On/Off**	Allows read and write access to bridge registers from the interconnect fabric using a specialized slave port. This option is required for **Requester/Completer** variants and optional for **Completer Only** variants. Enabling this option allows read and write access to bridge registers, except in the Completer-Only single dword variations.
Export MSI/MSI-X conduit interfaces	**On/Off**	When you turn this option on, the core exports top-level MSI and MSI-X interfaces that you can use to implement a Custom Interrupt Handler for MSI and MSI-X interrupts. For more information about the Custom Interrupt Handler, refer to *Interrupts for End Points Using the Avalon-MM Interface with Multiple MSI/MSI-X Support*. If you turn this option **Off**, the core handles interrupts internally. If you select this option, you must design your own external descriptor controller. The embedded controller does not support MSI-X.
Enable PCIe interrupt at power-on	**On/Off**	When you turn this option on, the Avalon-MM Arria 10 Hard IP for PCI Express enables the interrupt register at power-up. Turning off this option disables the interrupt register at power-up. The setting does not affect run-time configuration of the interrupt enable register. For the Avalon-MM interface with DMA, this value must be off.
Enable Hard IP Status Bus when using the AVMM interface	**On/Off**	When you turn this option on, your top-level variant includes the following top-level signals: • Link status signals • ECC error signals • TX and RX parity error signals • Completion header and data signals, indicating the total number of Completion TLPs currently stored in the RX buffer.
Address width of accessible PCIe memory space	**20–64**	Specifies the size of the PCIe memory space. The value you specify sets the width of the TX slave address, `txs_address` for 64-bit addresses.
Number of address pages	**2, 4, 8, 16, 32, 64, 128, 256, 512**	Specifies the number of consecutive address pages in the PCI Express address domain. This parameter is only necessary for 32-bit addresses.

Parameter	Value	Description
Size of address pages	**4 KByte–4GByte**	Sets the size of the PCI Express system pages. All pages must be the same size. This parameter is only necessary for 32-bit addresses.

Related Information

coreclkout_hip on page 7-5

Base Address Register (BAR) Settings

You can configure up to six 32-bit BARs or three 64-bit BARs.

Table 3-3: BAR Registers

Parameter	Value	Description
Type	**Disabled** **64-bit prefetchable memory** **32-bit non-prefetchable memory** **32-bit prefetchable memory** **I/O address space**	Defining memory as prefetchable allows data in the region to be fetched ahead anticipating that the requestor may require more data from the same region than was originally requested. If you specify that a memory is prefetchable, it must have the following 2 attributes: • Reads do not have side effects • Write merging is allowed The **32-bit prefetchable memory** and **I/O address space** BARs are only available for the **Legacy Endpoint**.
Size	Not configurable	Specifies the memory size calculated from other parameters you enter.

Device Identification Registers

Table 3-4: Device ID Registers

The following table lists the default values of the read-only Device ID registers. You can use the parameter editor to change the values of these registers. Refer to *Type 0 Configuration Space Registers* for the layout of the Device Identification registers.

Register Name	Range	Default Value	Description
Vendor ID	16 bits	0x00000000	Sets the read-only value of the `Vendor ID` register. This parameter cannot be set to 0xFFFF, per the *PCI Express Specification*. Address offset: 0x000.
Device ID	16 bits	0x00000000	Sets the read-only value of the `Device ID` register. This register is only valid in the Type 0 (Endpoint) Configuration Space. Address offset: 0x000.
Revision ID	8 bits	0x00000000	Sets the read-only value of the `Revision ID` register. Address offset: 0x008.
Class code	24 bits	0x00000000	Sets the read-only value of the `Class Code` register. Address offset: 0x008.
Subsystem Vendor ID	16 bits	0x00000000	Sets the read-only value of the `Subsystem Vendor ID` register in the PCI Type 0 Configuration Space. This parameter cannot be set to 0xFFFF per the *PCI Express Base Specification*. This value is assigned by PCI-SIG to the device manufacturer. This register is only valid in the Type 0 (Endpoint) Configuration Space. Address offset: 0x02C.
Subsystem Device ID	16 bits	0x00000000	Sets the read-only value of the `Subsystem Device ID` register in the PCI Type 0 Configuration Space. Address offset: 0x02C

Related Information

PCI Express Base Specification 3.0

PCI Express and PCI Capabilities Parameters

This group of parameters defines various capability properties of the IP core. Some of these parameters are stored in the PCI Configuration Space - PCI Compatible Configuration Space. The byte offset indicates the parameter address.

Device Capabilities

Table 3-5: Capabilities Registers

Parameter	Possible Values	Default Value	Description
Maximum payload size	**128 bytes** **256 bytes** **512 bytes** **1024 bytes** **2048 bytes**	128 bytes	Specifies the maximum payload size supported. This parameter sets the read-only value of the max payload size supported field of the Device Capabilities register (0x084[2:0]). Address: 0x084.
Completion timeout range	**ABCD** **BCD** **ABC** **AB** **B** **A** **None**	ABCD	Indicates device function support for the optional completion timeout programmability mechanism. This mechanism allows the system software to modify the completion timeout value. This field is applicable only to Root Ports and Endpoints that issue requests on their own behalf. Completion timeouts are specified and enabled in the Device Control 2 register (0x0A8) of the *PCI Express Capability Structure Version*. For all other functions this field is reserved and must be hardwired to 0x0000b. Four time value ranges are defined: • Range A: 50 us to 10 ms • Range B: 10 ms to 250 ms • Range C: 250 ms to 4 s • Range D: 4 s to 64 s Bits are set to show timeout value ranges supported. The function must implement a timeout value in the range 50 sto 50 ms. The following values specify the range: • None—Completion timeout programming is not supported • 0001 Range A • 0010 Range B • 0011 Ranges A and B • 0110 Ranges B and C

Parameter	Possible Values	Default Value	Description
			• 0111 Ranges A, B, and C • 1110 Ranges B, C and D • 1111 Ranges A, B, C, and D All other values are reserved. Altera recommends that the completion timeout mechanism expire in no less than 10 ms.
Disable completion timeout	**On/Off**	On	Disables the completion timeout mechanism. When **On**, the core supports the completion timeout disable mechanism via the PCI Express `Device Control Register 2`. The Application Layer logic must implement the actual completion timeout mechanism for the required ranges.

Error Reporting

Table 3-6: Error Reporting

Parameter	Value	Default Value	Description
Advanced error reporting (AER)	**On/Off**	Off	When **On**, enables the Advanced Error Reporting (AER) capability.
Enable ECRC checking	**On/Off**	Off	When **On**, enables ECRC checking. Sets the read-only value of the ECRC check capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability.
Enable ECRC generation	**On/Off**	Off	When **On**, enables ECRC generation capability. Sets the read-only value of the ECRC generation capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability.
Enable ECRC forwarding on the Avalon-ST interface	**On/Off**	Off	When **On**, enables ECRC forwarding to the Application Layer. On the Avalon-ST RX path, the incoming TLP contains the ECRC dword(1) and the `TD` bit is set if an ECRC exists. On the transmit the TLP from the Application Layer must contain the ECRC dword and have the `TD` bit set. Not applicable for Avalon-MM or Avalon-MM DMA interfaces.

Parameter	Value	Default Value	Description
Track RX completion buffer overflow on the Avalon-ST interface	**On/Off**	Off	When **On**, the core includes the `rxfx_cplbuf_ovf` output status signal to track the RX posted completion buffer overflow status. Not applicable for Avalon-MM or Avalon-MM DMA interfaces.

Note:

1. Throughout this user guide, the terms word, dword and qword have the same meaning that they have in the *PCI Express Base Specification*. A word is 16 bits, a dword is 32 bits, and a qword is 64 bits.

Link Capabilities

Table 3-7: Link Capabilities

Parameter	Value	Description
Link port number	**0x01**	Sets the read-only value of the port number field in the `Link Capabilities` register.
Data link layer active reporting	**On/Off**	Turn **On** this parameter for a downstream port, if the component supports the optional capability of reporting the DL_Active state of the Data Link Control and Management State Machine. For a hot-plug capable downstream port (as indicated by the `Hot Plug Capable` field of the `Slot Capabilities` register), this parameter must be turned **On**. For upstream ports and components that do not support this optional capability, turn **Off** this option. This parameter is only supported for the Arria 10 Hard IP for PCI Express in Root Port mode. Not applicable for Avalon-MM or Avalon-MM DMA interfaces.
Surprise down reporting	**On/Off**	When this option is **On**, a downstream port supports the optional capability of detecting and reporting the surprise down error condition. This parameter is only supported for the Arria 10 Hard IP for PCI Express in Root Port mode. Not applicable for Avalon-MM or Avalon-MM DMA interfaces.

Parameter	Value	Description
Slot clock configuration	**On/Off**	When **On**, indicates that the Endpoint or Root Port uses the same physical reference clock that the system provides on the connector. When **Off**, the IP core uses an independent clock regardless of the presence of a reference clock on the connector.

MSI and MSI-X Capabilities

Table 3-8: MSI and MSI-X Capabilities

Parameter	Value	Description
MSI messages requested	**1, 2, 4, 8, 16, 32**	Specifies the number of messages the Application Layer can request. Sets the value of the `Multiple Message Capable` field of the `Message Control` register, 0x050[31:16].
	MSI-X Capabilities	
Implement MSI-X	**On/Off**	When **On**, enables the MSI-X functionality.
	Bit Range	
Table size	[10:0]	System software reads this field to determine the MSI-X Table size <*n*>, which is encoded as <*n–1*>. For example, a returned value of 2047 indicates a table size of 2048. This field is read-only. Legal range is 0–2047 (2^{11}). Address offset: 0x068[26:16]
Table Offset	[31:0]	Points to the base of the MSI-X Table. The lower 3 bits of the table BAR indicator (BIR) are set to zero by software to form a 32-bit qword-aligned offset (1). This field is read-only.
Table BAR Indicator	[2:0]	Specifies which one of a function's BARs, located beginning at 0x10 in Configuration Space, is used to map the MSI-X table into memory space. This field is read-only. Legal range is 0–5.
Pending Bit Array (PBA) Offset	[31:0]	Used as an offset from the address contained in one of the function's Base Address registers to point to the base of the MSI-X PBA. The lower 3 bits of the PBA BIR are set to zero by software to form a 32-bit qword-aligned offset. This field is read-only.

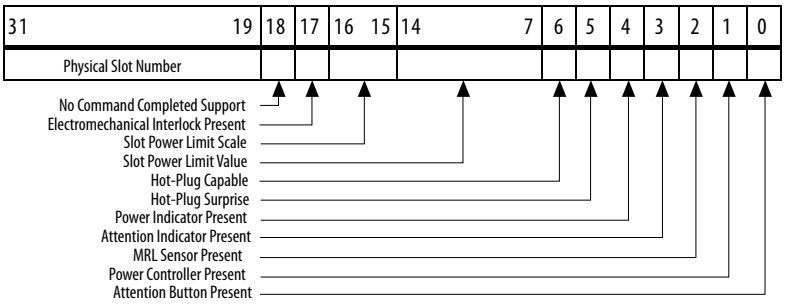
Parameter	Value	Description
Pending BAR Indicator	[2:0]	Specifies the function Base Address registers, located beginning at 0x10 in Configuration Space, that maps the MSI-X PBA into memory space. This field is read-only. Legal range is 0–5.

Note:

1. Throughout this user guide, the terms word, dword and qword have the same meaning that they have in the *PCI Express Base Specification*. A word is 16 bits, a dword is 32 bits, and a qword is 64 bits.

Slot Capabilities

Table 3-9: Slot Capabilities

Parameter	Value	Description
Use Slot register	**On/Off**	The slot capability is required for Root Ports if a slot is implemented on the port. Slot status is recorded in the `PCI Express Capabilities` register. This parameter is only supported in Root Port mode. Defines the characteristics of the slot. You turn on this option by selecting **Enable slot capability**. The various bits are defined as follows: 
Slot power scale	`0–3`	Specifies the scale used for the **Slot power limit**. The following coefficients are defined: • 0 = 1.0x • 1 = 0.1x • 2 = 0.01x • 3 = 0.001x The default value prior to hardware and firmware initialization is b'00. Writes to this register also cause the port to send the `Set_Slot_Power_Limit` Message. Refer to Section 6.9 of the *PCI Express Base Specification Revision* for more information.

Parameter	Value	Description
Slot power limit	0–255	In combination with the **Slot power scale value**, specifies the upper limit in watts on power supplied by the slot. Refer to Section 7.8.9 of the *PCI Express Base Specification* for more information.
Slot number	0-8191	Specifies the slot number.

Power Management

Table 3-10: Power Management Parameters

Parameter	Value	Description
Endpoint L0s acceptable latency	**Maximum of 64 ns** **Maximum of 128 ns** **Maximum of 256 ns** **Maximum of 512 ns** **Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **No limit**	This design parameter specifies the maximum acceptable latency that the device can tolerate to exit the L0s state for any links between the device and the root complex. It sets the read-only value of the Endpoint L0s acceptable latency field of the `Device Capabilities Register` (0x084). This Endpoint does not support the L0s or L1 states. However, in a switched system there may be links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 64 ns. This is the safest setting for most designs.
Endpoint L1 acceptable latency	**Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **Maximum of 8 us** **Maximum of 16 us** **Maximum of 32 us** **No limit**	This value indicates the acceptable latency that an Endpoint can withstand in the transition from the L1 to L0 state. It is an indirect measure of the Endpoint's internal buffering. It sets the read-only value of the Endpoint L1 acceptable latency field of the `Device Capabilities Register`. This Endpoint does not support the L0s or L1 states. However, a switched system may include links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 1 µs. This is the safest setting for most designs.

Physical Layout of Hard IP In Arria 10 Devices

4

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

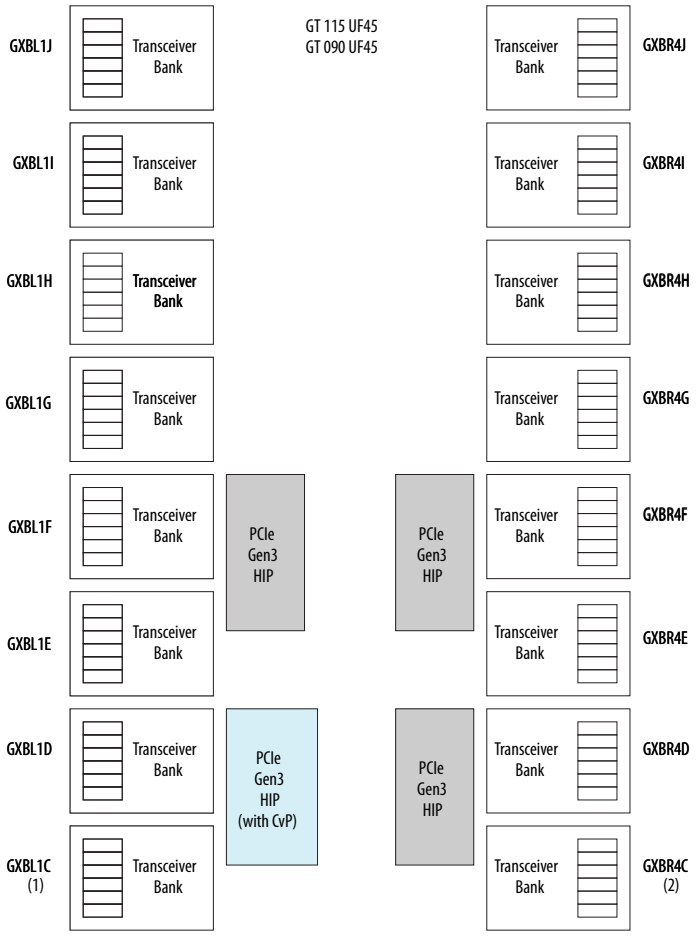
Arria 10 devices include 1–4 hard IP blocks for PCI Express. The bottom left hard IP block includes the CvP functionality for flip chip packages. For other package types, the CvP functionality is in the bottom right block.

Note: Arria 10 devices do not support configurations that configure a bottom (left or right) hard IP block with a Gen3 x4 or Gen3 x8 IP core and also configure the top hard IP block on the same side with a Gen3 x1 or Gen3 x2 IP core variation.

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

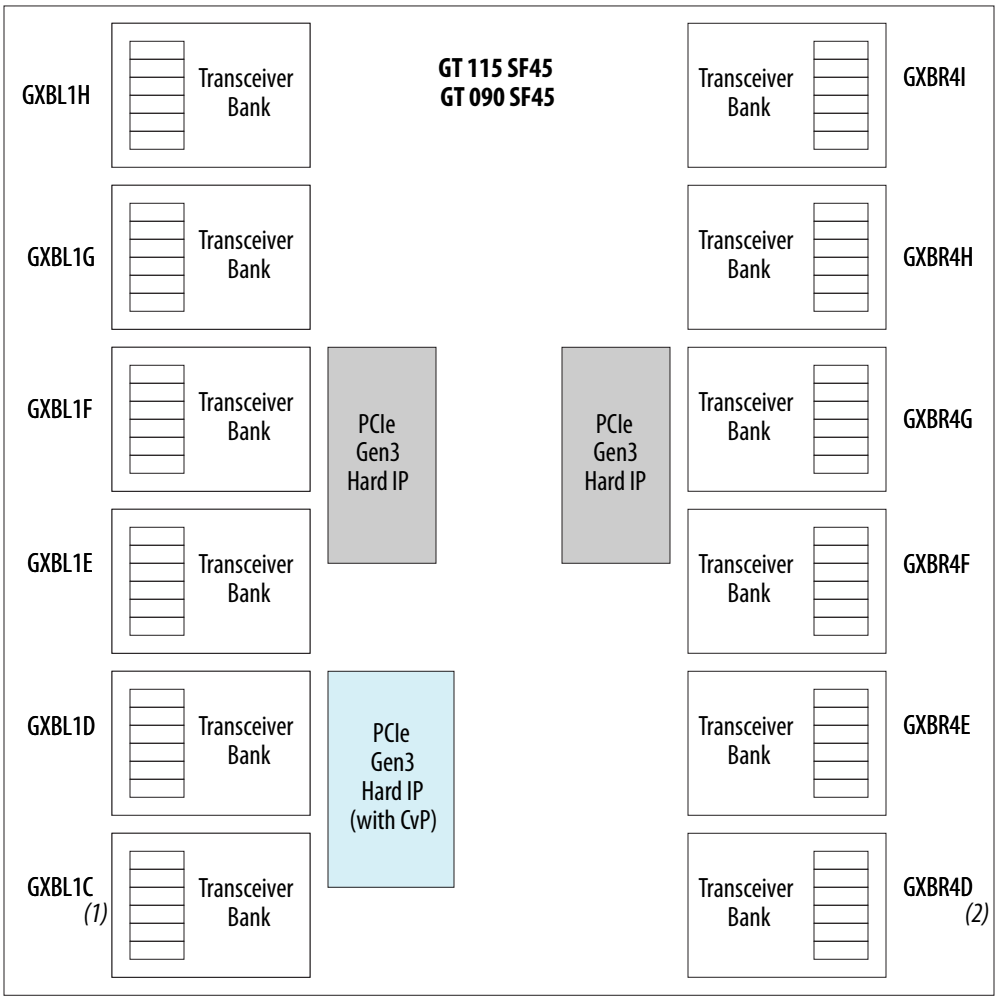
ISO 9001:2008 Registered

Figure 4-1: Arria 10 Devices with 96 Transceiver Channels and Four PCIe Hard IP Blocks



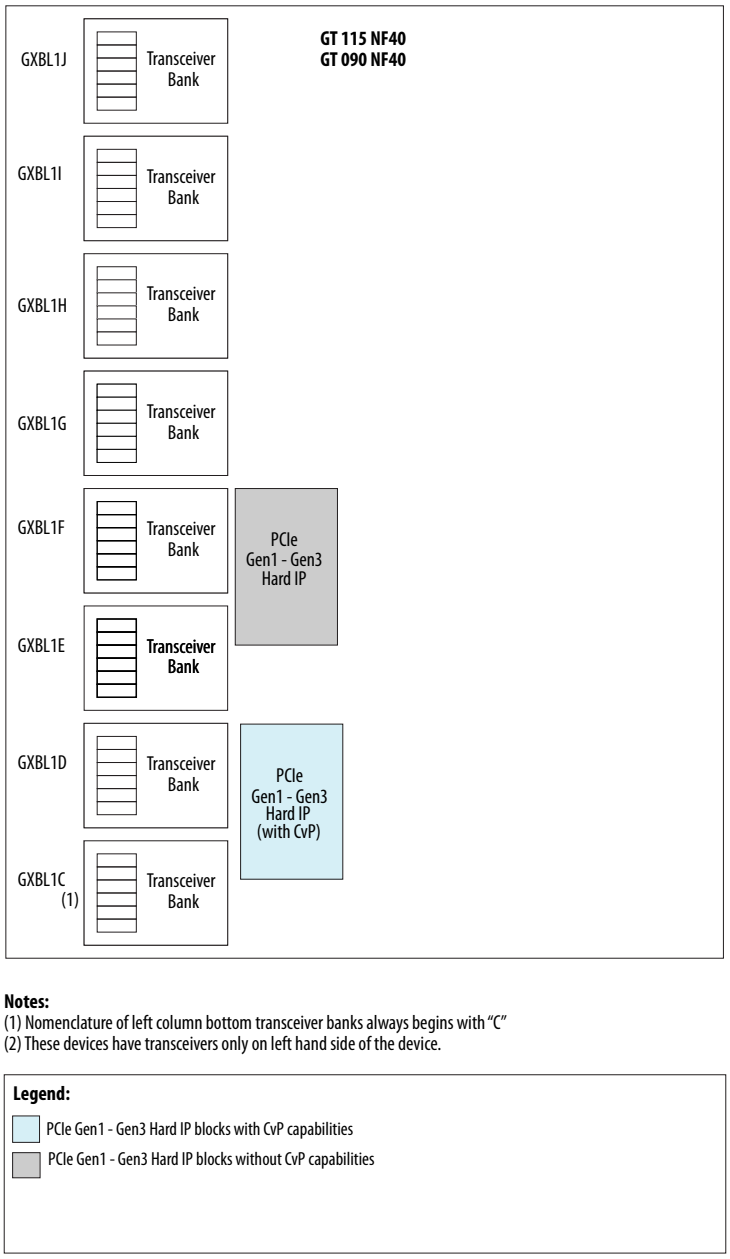
Notes:
(1) Nomenclature of left column bottom transceiver banks always begins with "C"
(2) Nomenclature of right column bottom transceiver banks may begin with "C", "D", or "E".

Figure 4-2: Arria 10 Devices with 72 Transceiver Channels and Four PCIe Hard IP Blocks



Notes:
(1) Nomenclature of left column bottom transceiver banks always begins with "C"
(2) Nomenclature of right column bottom transceiver banks may begin with "C", "D", or "E".

Figure 4-3: Arria 10 GT Devices with 48 Transceiver Channels and Two PCIe Hard IP Blocks



Refer to the *Arria 10 Transceiver Layout* in the *Arria 10 Transceiver PHY User Guide* for comprehensive figures for Arria 10 GT, GX, and SX devices.

Related Information

Arria 10 Transceiver PHY User Guide

Channel and Pin Placement for the Gen1, Gen2, and Gen3 Data Rates

The following figures illustrate the x1, x2, x4, and x8 channel and pin placements for the Arria 10 Hard IP for PCI Express.

In these figures, channels that are not used for the PCI Express protocol are available for other protocols. Unused channels are shown in gray.

Note: In all configurations, physical channel 4 in the PCS connects to logical channel 0 in the hard IP. You cannot change the channel placements illustrated below.

For the possible values of <txvr_block_N> and <txvr_block_N+1>, refer to the figures that show the physical location of the Hard IP PCIe blocks in the different types of Arria 10 devices, at the start of this chapter. For each HIP block, the transceiver block that is adjacent and extends below the HIP block, is <txvr_block_N>, and the transceiver block that is directly above <txvr_block_N> is <txvr_block_N+1>. For example, in an Arria 10 device with 96 transceiver channels and four PCIe HIP blocks, if your design uses the HIP block that supports CvP, <txvr_block_N> is GXB1C and <txvr_block_N+1> is GXB1D.

Figure 4-4: Arria 10 Gen1, Gen2, and Gen3 x1 Channel and Pin Placement

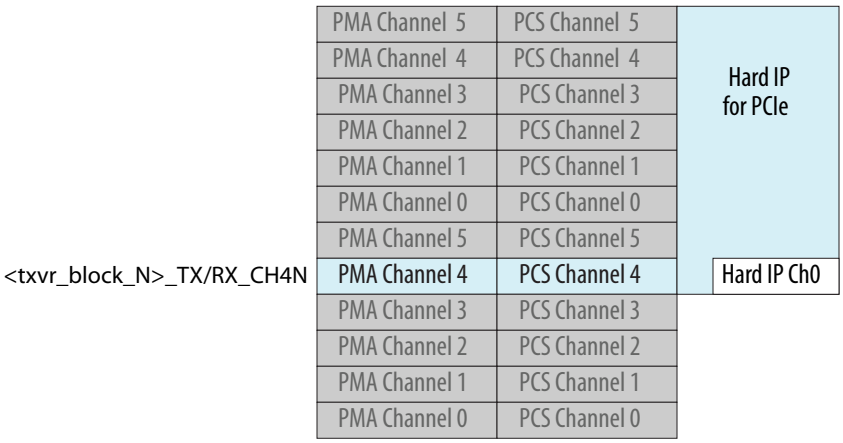


Figure 4-5: Arria 10 Gen1 Gen2, and Gen3 x2 Channel and Pin Placement

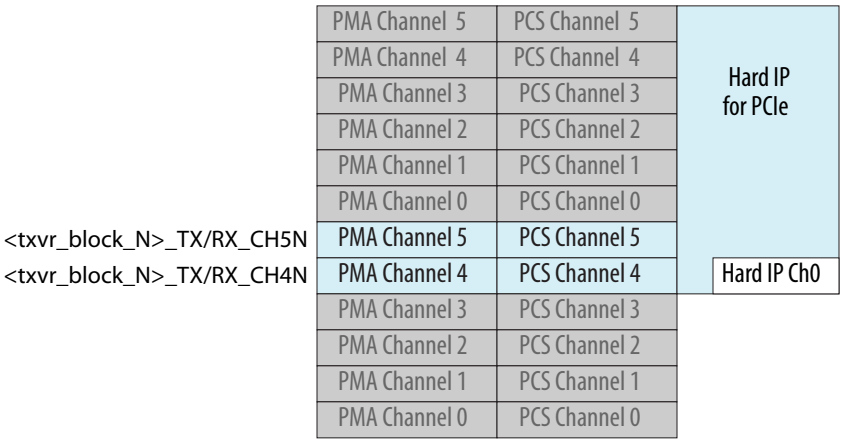


Figure 4-6: Arria 10 Gen1, Gen2, and Gen3 x4 Channel and Pin Placement

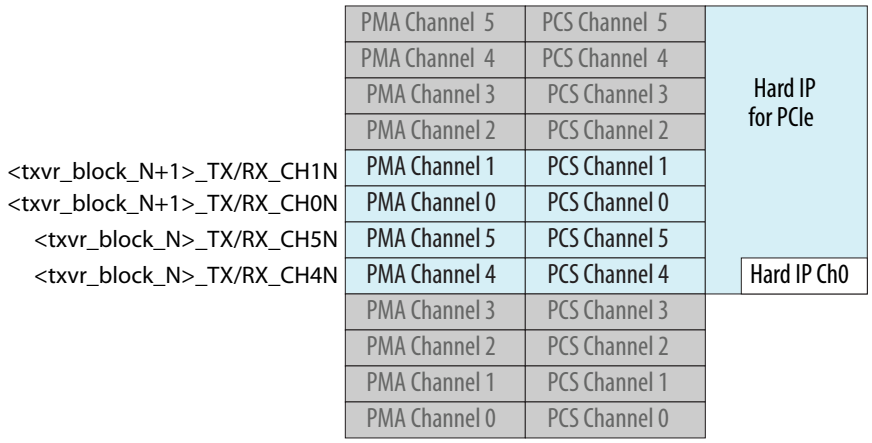
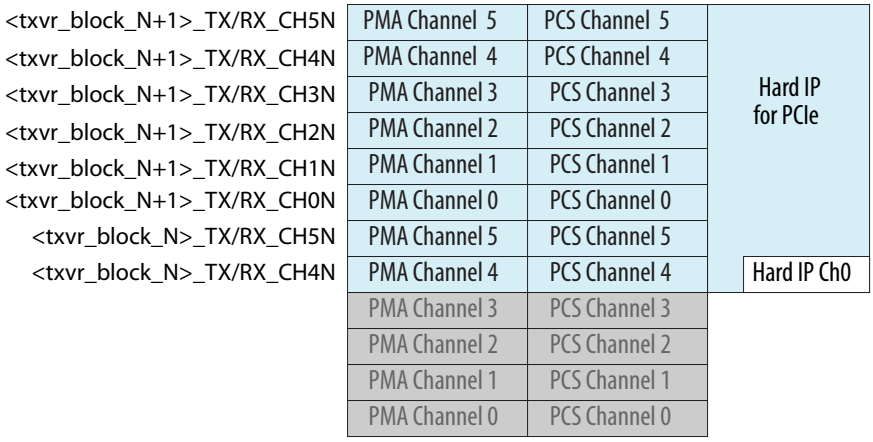


Figure 4-7: Arria 10 Gen1, Gen2, and Gen3 x8 Channel and Pin Placement



Channel Placement and fPLL Usage for the Gen1 and Gen2 Data Rates

The following figures illustrate the x1, x2, x4, and x8 channel placement for the Arria 10 Hard IP for PCI Express. In these figures, channels that are not used for the PCI Express protocol are available for other protocols. Unused channels are shown in gray.

Note: In all configurations, physical channel 4 in the PCS connects to logical channel 0 in the hard IP. You cannot change the channel placements illustrated below.

Figure 4-8: Arria 10 Gen1 and Gen2 x1 Channel Placement

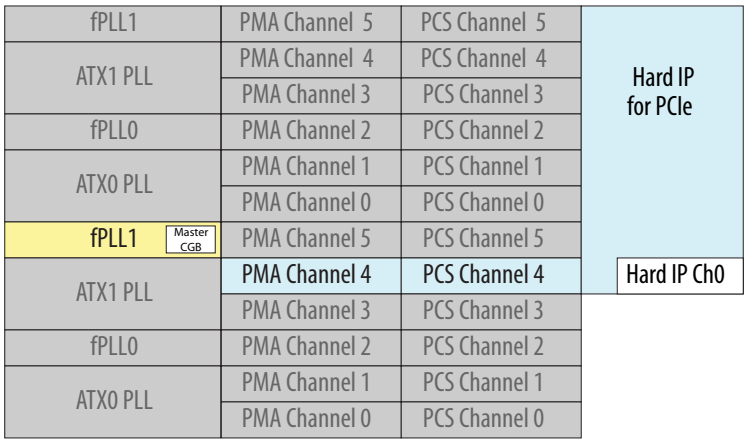


Figure 4-9: Arria 10 Gen1 and Gen2 x2 Channel Placement

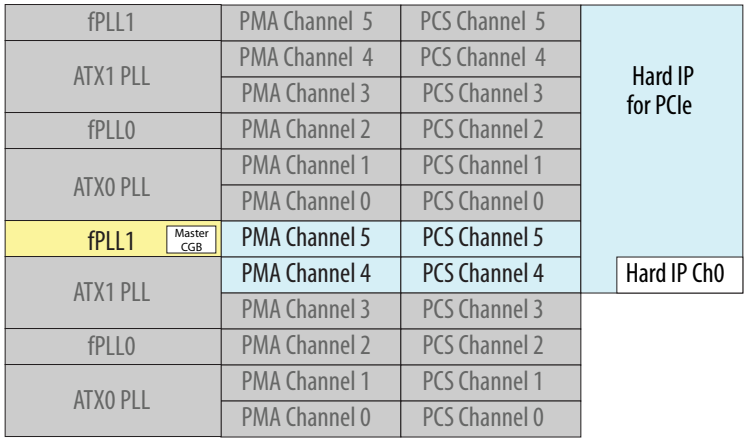


Figure 4-10: Arria 10 Gen1 and Gen2 x4 Channel Placement

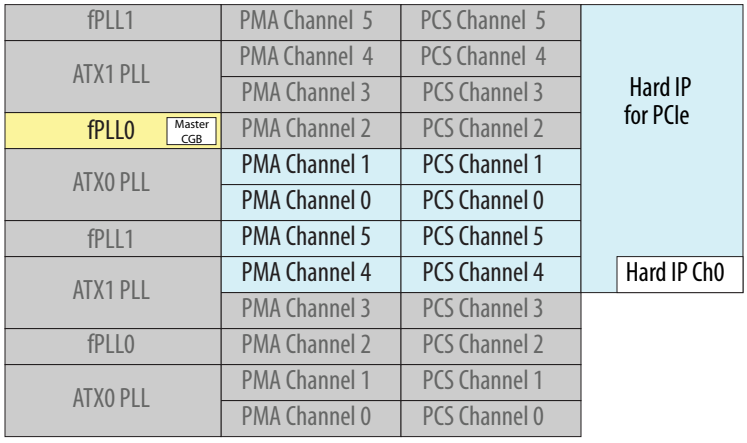
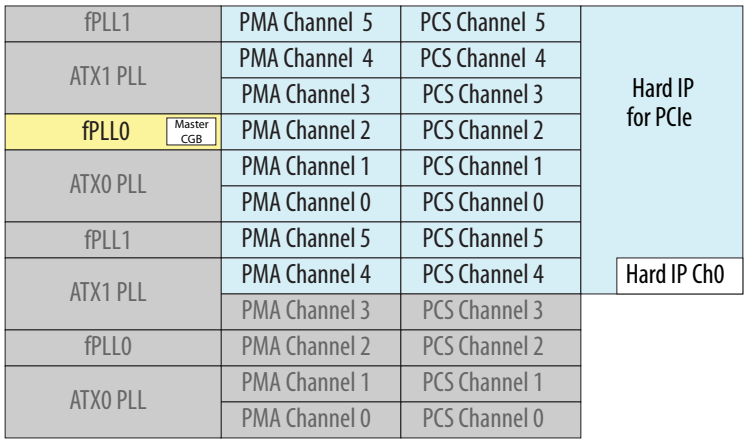


Figure 4-11: Gen1 and Gen2 x8 Channel Placement



Channel Placement and fPLL and ATX PLL Usage for the Gen3 Data Rate

The following figures illustrate the x1, x2, x4, and x8 channel placement for the Arria 10 Hard IP for PCI Express. Gen3 variants must initially train at the Gen1 data rate. Consequently, Gen3 variants require an fPLL to generate the 2.5 and 5.0 Gbps clocks, and an ATX PLL to generate the 8.0 Gbps clock.

In these figures, channels that are not used for the PCI Express protocol are available for other protocols. Unused channels are shown in gray.

Note: In all configurations, physical channel 4 in the PCS connects to logical channel 0 in the hard IP. You cannot change the channel placements illustrated below.

Figure 4-12: Arria 10 Gen3 x1 Channel Placement

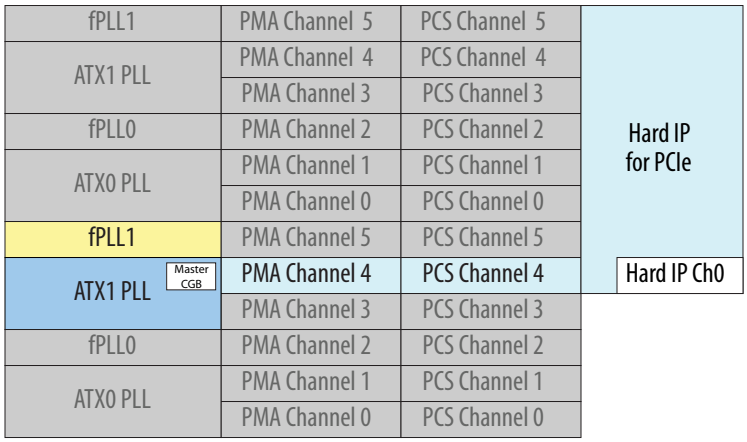


Figure 4-13: Arria 10 Gen3 x2 Channel Placement

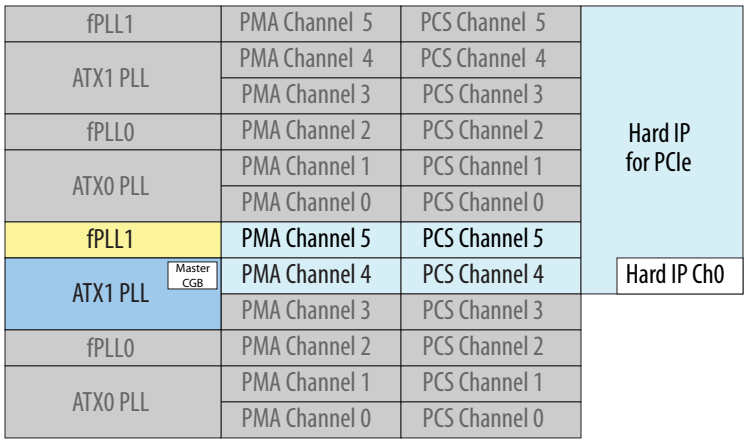


Figure 4-14: Arria 10 Gen3 x4 Channel Placement

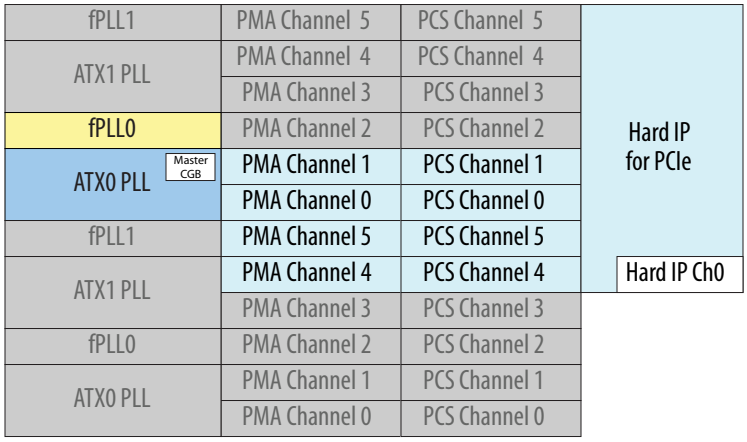
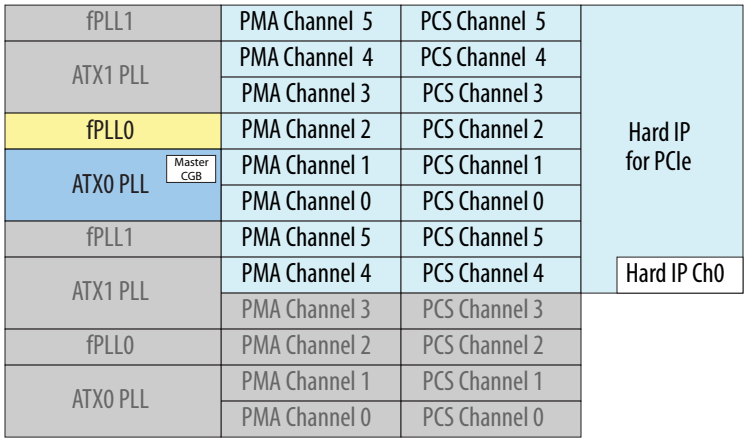


Figure 4-15: Gen3 x8 Channel Placement



64- or 128-Bit Avalon-MM Interface to the Application Layer

5

2015.06.05

UG-01145_avmm

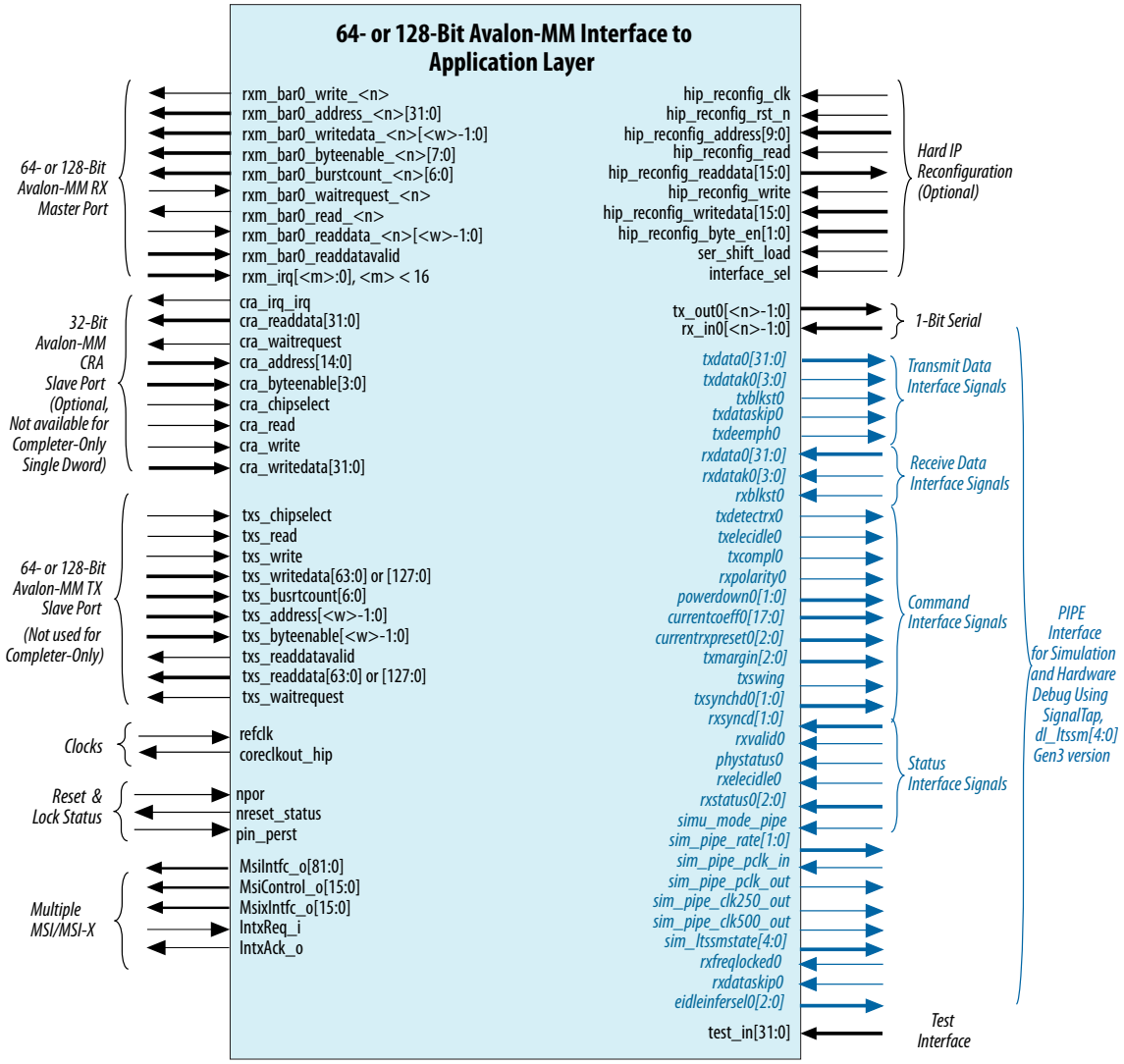
Subscribe Send Feedback

This chapter describes the top-level signals of the Arria 10 Hard IP for PCI Express using the Avalon-MM interface to the Application Layer. The Avalon-MM bridge translates PCI Express read, write and completion TLPs into standard Avalon-MM read and write commands for the Avalon-MM RX Master Port interface. For the Avalon-MM TX Slave Port interface, the bridge translates Avalon-MM reads and writes into PCI Express TLPs. The Avalon-MM read and write commands are the same as those used by master and slave interfaces to access memories and registers. Consequently, you do not need a detailed understanding of the PCI Express TLPs to use this Avalon-MM variant.

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure 5-1: Signals in 64- or 128-Bit Avalon-MM Interface to the Application Layer



Note: Signals listed for BAR0 are the same as those for BAR1–BAR5 when those BARs are enabled in the parameter editor.

Variations using the Avalon-MM interface implement the Avalon-MM protocol described in the *Avalon Interface Specifications*. Refer to this specification for information about the Avalon-MM protocol, including timing diagrams.

Related Information

Avalon Interface Specifications

32-Bit Non-Bursting Avalon-MM Control Register Access (CRA) Slave Signals

The optional CRA port for the full-featured IP core allows upstream PCI Express devices and external Avalon-MM masters to access internal control and status registers.

Table 5-1: Avalon-MM CRA Slave Interface Signals

Signal Name	Direction	Description
CraIrq_o	Output	Interrupt request. A port request for an Avalon-MM interrupt.
CraReadData_o[31:0]	Output	Read data lines
CraWaitRequest_o	Output	Wait request to hold off more requests
CraAddress_i[13:0]	Input	An address space of 16,384 bytes is allocated for the control registers. Avalon-MM slave addresses provide address resolution down to the width of the slave data bus. Because all addresses are byte addresses, this address logically goes down to bit 2. Bits 1 and 0 are 0.
CraByteEnable_i[3:0]	Input	Byte enable
CraChipSelect_i	Input	Chip select signal to this slave
CraRead_i	Input	Read enable
CraWrite_i	Input	Write request
CraWriteData_i[31:0]	Input	Write data

RX Avalon-MM Master Signals

This Avalon-MM master port propagates PCI Express requests to the Qsys interconnect fabric. For the full-feature IP core it propagates requests as bursting reads or writes. A separate Avalon-MM master port corresponds to each BAR.

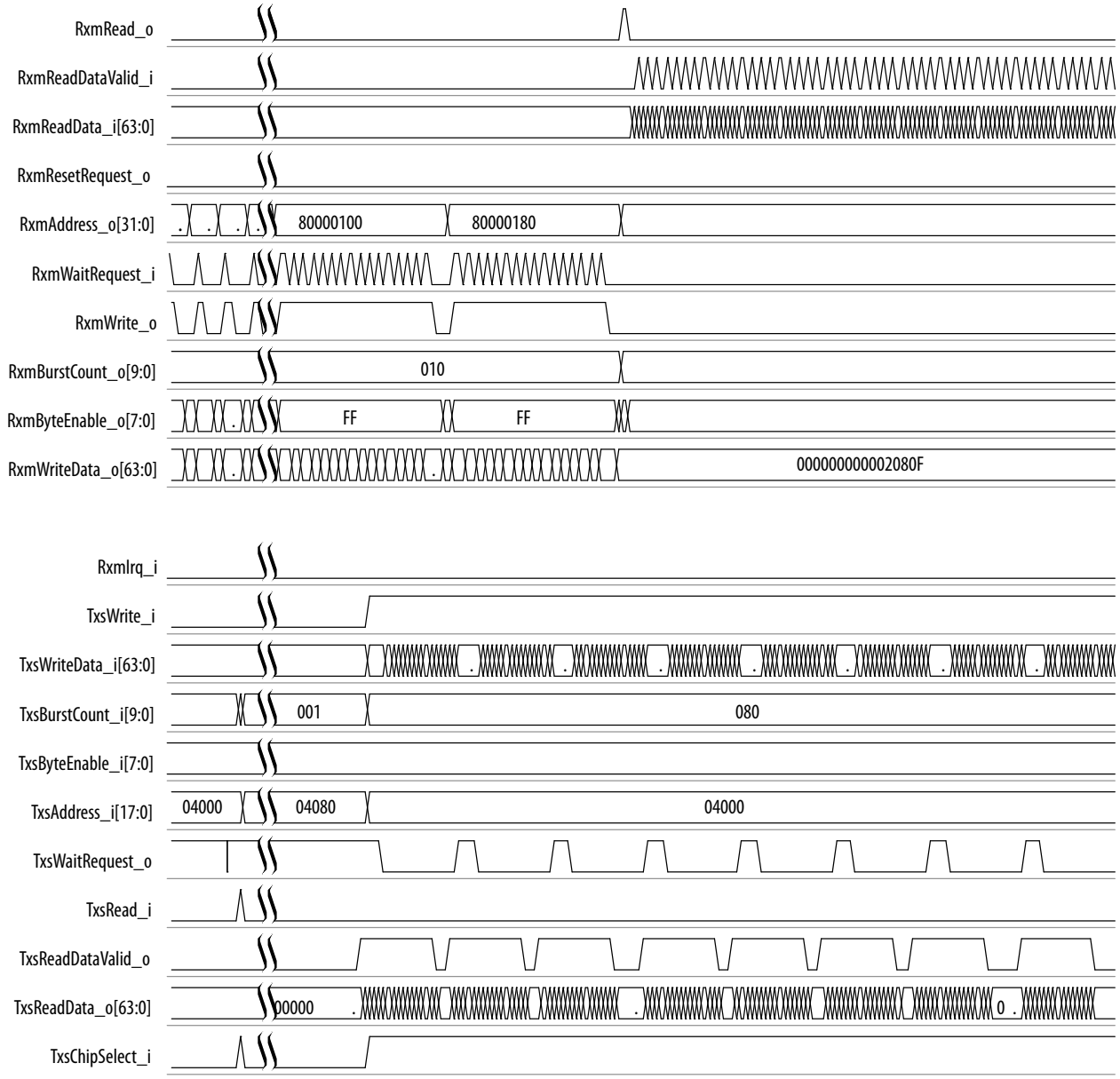
Table 5-2: Avalon-MM RX Master Interface Signals

Signals that include Bar number 0 also exist for BAR1–BAR5 when additional BARs are enabled.

Signal Name	Direction	Description
RxmWrite<n>	Output	Asserted by the core to request a write to an Avalon-MM slave.
RxmAddress_<*n*>_o[31:0]	Output	The address of the Avalon-MM slave being accessed.
RxmWriteData_<*n*>_o[<*w*>-1:0]	Output	RX data being written to slave. <*w*> = 64 or 128 for the full-featured IP core. <*w*> = 32 for the completer-only IP core.
RxmByteEnable_<*n*>_o[<*w*>-1:0]	Output	Byte enable for write data.
RXMBurstCount_<*n*>_o[6 or 5:0]	Output	The burst count, measured in qwords, of the RX write or read request. The width indicates the maximum data that can be requested. The maximum data in a burst is 512 bytes.
RXMWaitRequest_<*n*>_o	Input	Asserted by the external Avalon-MM slave to hold data transfer.
RXMRead_<*n*>_o	Output	Asserted by the core to request a read.
RXMReadData_<*n*>_o[<*w*>-1:0]	Input	Read data returned from Avalon-MM slave in response to a read request. This data is sent to the IP core through the TX interface. <*w*> = 64 or 128 for the full-featured IP core. <*w*> = 32 for the completer-only IP core.
RXMReadDataValid_<*n*>_i	Input	Asserted by the system interconnect fabric to indicate that the read data on is valid.
RxmIrq_<n>[<*m*>:0], <*m*>< 16	Input	Indicates an interrupt request asserted from the system interconnect fabric. This signal is only available when the CRA port is enabled. Qsys-generated variations have as many as 16 individual interrupt signals (<*m*>≤15). If rxm_irq_<n>[<*m*>:0] is asserted on consecutive cycles without the deassertion of all interrupt inputs, no MSI message is sent for subsequent interrupts. To avoid losing interrupts, software must ensure that all interrupt sources are cleared for each MSI message received.

The following figure illustrates the RX master port propagating requests to the Application Layer and also shows simultaneous, DMA read and write activity

Figure 5-2: Simultaneous DMA Read, DMA Write, and Target Access



64- or 128-Bit Bursting TX Avalon-MM Slave Signals

This optional Avalon-MM bursting slave port propagates requests from the interconnect fabric to the full-featured Avalon-MM Arria 10 Hard IP for PCI Express. Requests from the interconnect fabric are translated into PCI Express request packets. Incoming requests can be up to 512 bytes. For better performance, Altera recommends using smaller read request size (a maximum of 512 bytes).

Table 5-3: Avalon-MM TX Slave Interface Signals

Signal Name	Direction	Description
TxsChipSelect_i	Input	The system interconnect fabric asserts this signal to select the TX slave port.
TxsRead_i	Input	Read request asserted by the system interconnect fabric to request a read.
TxsWrite_i	Input	Write request asserted by the system interconnect fabric to request a write.
TxsWriteData[127 or 63:0]	Input	Write data sent by the external Avalon-MM master to the TX slave port.
TxsBurstCount[6 or 5:0]	Input	Asserted by the system interconnect fabric indicating the amount of data requested. The count unit is the amount of data that is transferred in a single cycle, that is, the width of the bus. The burst count is limited to 512 bytes.
TxsAddress_i[<w>-1:0]	Input	Address of the read or write request from the external Avalon-MM master. This address translates to 64-bit or 32-bit PCI Express addresses based on the translation table. The <*w*> value is determined when the system is created.

Signal Name	Direction	Description

TxsByteEnable_i[<w>-1:0]	Input	Write byte enable for data. A burst must be continuous. Therefore all intermediate data phases of a burst must have a byte enable value of 0xFF. The first and final data phases of a burst can have other valid values.
		For the 128-bit interface, the following restrictions apply:
		• All bytes of a single dword must either be enabled or disabled
		• If more than 1 dword is enabled, the enabled dwords must be contiguous. The following patterns are legal:
		  • 16'bF000
		  • 16'b0F00
		  • 16'b00F0
		  • 16'b000F
		  • 16'bFF00
		  • 16'b0FF0
		  • 16'b00FF
		  • 16'bFFF0
		  • 16'b0FFF
		  • 16'bFFFF

TxsReadDataValid_o	Output	Asserted by the bridge to indicate that read data is valid.
TxsReadData_o[127 or 63:0]	Output	The bridge returns the read data on this bus when the RX read completions for the read have been received and stored in the internal buffer.
TxsWaitrequest_o	Output	Asserted by the bridge to hold off read or write data when running out of buffer space. If this signal is asserted during an operation, the master should maintain the TxsRead_i signal (or TxsWrite_i signal and TxsWriteData) stable until after TxsWaitrequest_o is deasserted. txs_Read must be deasserted when TxsWaitrequest_o is deasserted.

Clock Signals

Table 5-4: Clock Signals

Signal	Direction	Description
refclk	Input	Reference clock for the IP core. It must have the frequency specified under the **System Settings** heading in the parameter editor. This is a dedicated free running input clock to the dedicated REFCLK pin.
coreclkout_hip	Output	This is a fixed frequency clock used by the Data Link and Transaction Layers.

Related Information

Clocks on page 7-4

Reset

Refer to *Reset and Clocks* for more information about the reset sequence and a block diagram of the reset logic.

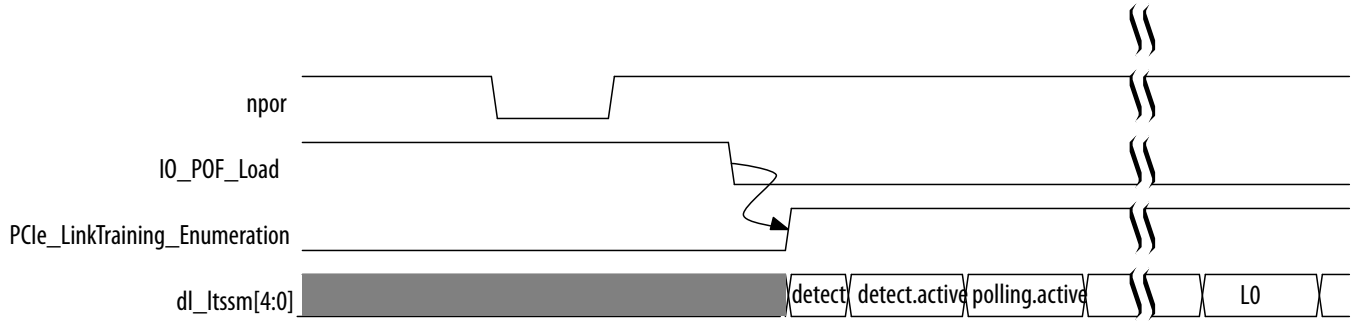
Table 5-5: Reset Signals

Signal	Direction	Description
npor	Input	Active low reset signal. In the Altera hardware example designs, npor is the OR of pin_perst and local_rstn coming from the software Application Layer. If you do not drive a soft reset signal from the Application Layer, this signal must be derived from pin_perst. You cannot disable this signal. Resets the entire IP Core and transceiver. Asynchronous. This signal is *edge, not level* sensitive; consequently, you cannot use a low value on this signal to hold custom logic in reset. For more information about the reset controller, refer to *Reset*.
nreset_status	Output	Active low reset signal. It is derived from npor or pin_perstn. You can use this signal to reset the Application Layer.
pin_perst	Input	Active low reset from the PCIe reset pin of the device. pin_perst resets the datapath and control registers. Configuration via Protocol (CvP) requires this signal. For more information about CvP refer to *Configuration via Protocol (CvP)*. Arria 10 devices can have up to 4 instances of the Hard IP for PCI Express. Each instance has its own pin_perst signal. *You*

Signal	Direction	Description
		must connect the `pin_perst` *of each Hard IP instance to the corresponding* `nPERST` *pin of the device.* These pins have the following locations: • `NPERSTL0`: bottom left Hard IP and CvP blocks • `NPERSTL1`: top left Hard IP block • `NPERSTR0`: bottom right Hard IP block • `NPERSTR1`: top right Hard IP block For example, if you are using the Hard IP instance in the bottom left corner of the device, you must connect `pin_perst` to `NPERSL0`. For maximum use of the Arria 10 device, Altera recommends that you use the bottom left Hard IP first. This is the only location that supports CvP over a PCIe link. If your design does not require CvP, you may select other Hard IP blocks. Refer to the appropriate device pinout for correct pin assignment for more detailed information about these pins. The *PCI Express Card Electromechanical Specification 2.0* specifies this pin requires 3.3 V. You can drive this 3.3V signal to the `nPERST*` even if the V_{CCPGM} of the bank is not 3.3V if the following 2 conditions are met: • The input signal meets the V_{IH} and V_{IL} specification for LVTTL. • The input signal meets the overshoot specification for 100°C operation as defined in the device handbook.

Figure 5-3: Reset and Link Training Timing Relationships

The following figure illustrates the timing relationship between `npor` and the LTSSM L0 state.



Note: To meet the 100 ms system configuration time, you must use the fast passive parallel configuration scheme with CvP and a 32-bit data width (FPP x32) or use the CvP in autonomous mode.

Related Information

- PCI Express Card Electromechanical Specification 2.0
- Configuration via Protocol (CvP) Implementation in Altera FPGAs User Guide

Interrupts for Endpoints when Multiple MSI/MSI-X Support Is Enabled

Table 5-6: Exported Interrupt Signals for Endpoints when Multiple MSI/MSI-X Support is Enabled

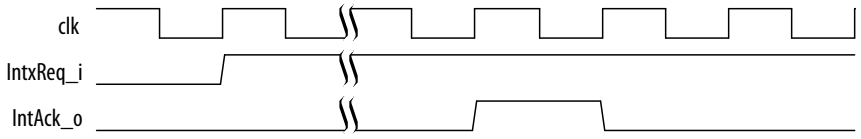
The following table describes the IP core's exported interrupt signals when you turn on **Enable multiple MSI/MSI-X support** under the **Avalon-MM System Settings** banner in the parameter editor.

Signal	Direction	Description
`MsiIntfc_o[81:0]`	Output	This bus provides the following MSI address, data, and enabled signals: • `MsiIntf_o[81]`: Master enable • `MsiIntf_o[80}`: MSI enable • `MsiIntf_o[79:64]`: MSI data • `MsiIntf_o[63:0]`: MSI address
`MsiControl_o[15:0]`	Output	Provides for system software control of MSI as defined in Section 6.8.1.3 *Message Control for MSI* in the *PCI Local Bus Specification, Rev. 3.0*. The following fields are defined: • `MsiControl_o[15:9]`: Reserved • `MsiControl_o[8]`: Per-vector masking capable • `MsiControl_o[7]`: 64-bit address capable • `MsiControl_o[6:4]`: Multiple Message Enable • `MsiControl_o[3:1]`: MSI Message Capable • `MsiControl_o[0]`: MSI Enable.
`MsixIntfc_o[15:0]`	Output	Provides for system software control of MSI-X as defined in Section 6.8.2.3 *Message Control for MSI-X* in the *PCI Local Bus Specification, Rev. 3.0*. The following fields are defined: • `MsixIntfc_o[15]`: Enable • `MsixIntfc_o[14]`: Mask • `MsixIntfc_o[13:11]`: Reserved • `MsixIntfc_o[10:0]`: Table size
`IntxReq_i`	Input	When asserted, the Endpoint is requesting attention from the interrupt service routine unless MSI or MSI-X interrupts are enabled. Remains asserted until the device driver clears the pending request.

Signal	Direction	Description
IntxAck_o	Output	This signal is the acknowledge for IntxReq_i. It is asserted for at least one cycle either when either of the following events occur: • The Assert_INTA message TLP has been transmitted in response to the assertion of the IntxReq_i. • The Deassert_INTA message TLP has been transmitted in response to the deassertion of the IntxReq_i signal. Refer to the timing diagrams below.

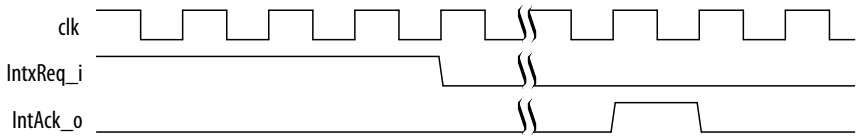
The following figure illustrates interrupt timing for the legacy interface. In this figure the assertion of IntxReq_i instructs the Hard IP for PCI Express to send an Assert_INTA message TLP.

Figure 5-4: Legacy Interrupt Assertion



The following figure illustrates the timing for deassertion of legacy interrupts. The assertion of IntxReq_i instructs the Hard IP for PCI Express to send a Deassert_INTA message.

Figure 5-5: Legacy Interrupt Deassertion



Hard IP Reconfiguration Interface

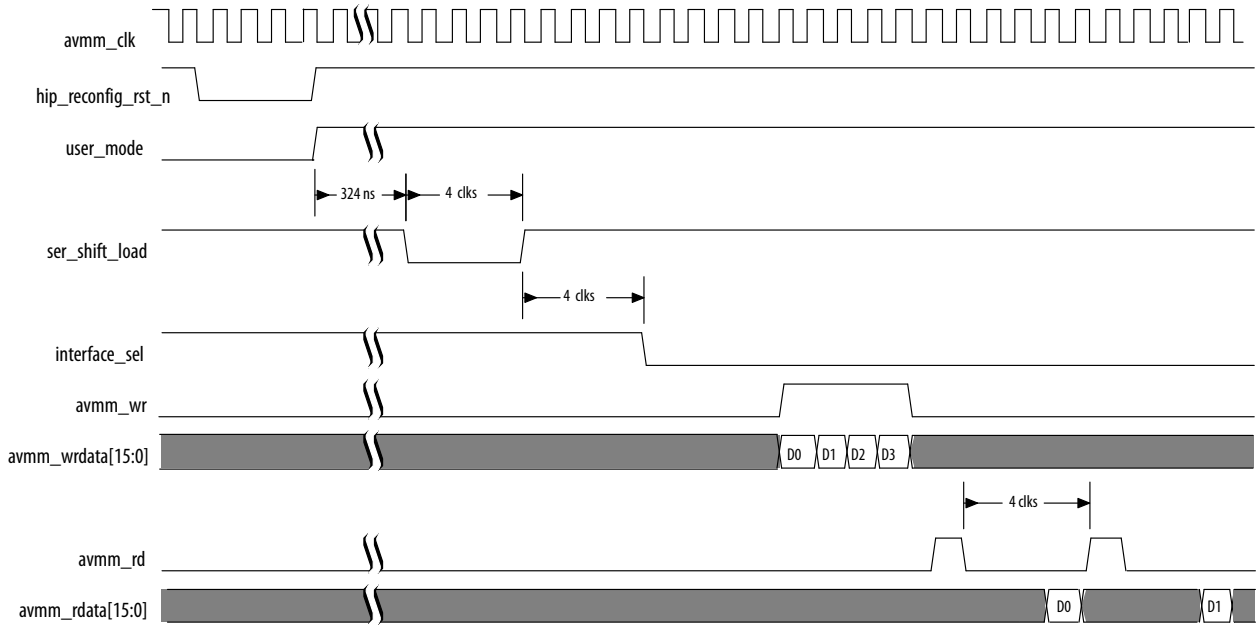
The Hard IP reconfiguration interface is an Avalon-MM slave interface with a 10-bit address and 16-bit data bus. You can use this bus to dynamically modify the value of configuration registers that are read-only at run time. To ensure proper system operation, reset or repeat device enumeration of the PCI Express link after changing the value of read-only configuration registers of the Hard IP.

Table 5-7: Hard IP Reconfiguration Signals

Signal	Direction	Description
hip_reconfig_clk	Input	Reconfiguration clock. The frequency range for this clock is 100–125 MHz.

Signal	Direction	Description
hip_reconfig_rst_n	Input	Active-low Avalon-MM reset. Resets all of the dynamic reconfiguration registers to their default values as described in *Hard IP Reconfiguration Registers*.
hip_reconfig_address[9:0]	Input	The 10-bit reconfiguration address.
hip_reconfig_read	Input	Read signal. This interface is not pipelined. You must wait for the return of the hip_reconfig_readdata[15:0] from the current read before starting another read operation.
hip_reconfig_readdata[15:0]	Output	16-bit read data. hip_reconfig_readdata[15:0] is valid on the third cycle after the assertion of hip_reconfig_read.
hip_reconfig_write	Input	Write signal.
hip_reconfig_writedata[15:0]	Input	16-bit write model.
hip_reconfig_byte_en[1:0]	Input	Byte enables, currently unused.
ser_shift_load	Input	You must toggle this signal once after changing to user mode before the first access to read-only registers. This signal should remain asserted for a minimum of 324 ns after switching to user mode.
interface_sel	Input	A selector which must be asserted when performing dynamic reconfiguration. Drive this signal low 4 clock cycles after the release of ser_shif t_load.

Figure 5-6: Hard IP Reconfiguration Bus Timing of Read-Only Registers



For a detailed description of the Avalon-MM protocol, refer to the *Avalon Memory Mapped Interfaces* chapter in the *Avalon Interface Specifications*.

Related Information

Avalon Interface Specifications

Physical Layer Interface Signals

Altera provides an integrated solution with the Transaction, Data Link and Physical Layers. The IP Parameter Editor generates a SERDES variation file, **<*variation*>_serdes.v** or **.vhd** , in addition to the Hard IP variation file, **<*variation*>.v** or **.vhd**. The SERDES entity is included in the library files for PCI Express.

Serial Data Signals

Table 5-8: 1-Bit Interface Signals

Signal	Direction	Description
`tx_out[7:0]` (1)	Output	Transmit output. These signals are the serial outputs of lanes 7–0.
`rx_in[7:0]` (1)	Input	Receive input. These signals are the serial inputs of lanes 7–0.

Note:

1. The x1 IP core only has lane 0. The x2 IP core only has lanes 1–0. The x4 IP core only has lanes 3–0.

Refer to *Pin-out Files for Altera Devices* for pin-out tables for all Altera devices in **.pdf**, **.txt**, and **.xls** formats.

Transceiver channels are arranged in groups of six. For GX devices, the lowest six channels on the left side of the device are labeled GXB_L0, the next group is GXB_L1, and so on. Channels on the right side of the device are labeled GXB_R0, GXB_R1, and so on. Be sure to connect the Hard IP for PCI Express on the left side of the device to appropriate channels on the left side of the device, as specified in the *Pin-out Files for Altera Devices*.

Related Information

- **Physical Layout of Hard IP In Arria 10 Devices** on page 4-1
- **Pin-out Files for Altera Devices**

PIPE Interface Signals

These PIPE signals are available for Gen1, Gen2, and Gen3 variants so that you can simulate using either the serial or the PIPE interface. Simulation is much faster using the PIPE interface because the PIPE simulation bypasses the SERDES model . By default, the PIPE interface is 8 bits for Gen1 and Gen2 and 32 bits for Gen3. You can use the PIPE interface for simulation even though your actual design includes a serial interface to the internal transceivers. However, it is not possible to use the Hard IP PIPE interface in hardware, including probing these signals using SignalTap® II Embedded Logic Analyzer.

Note: The Altera Root Port BFM bypasses Gen3 Phase 2 and Phase 3 Equalization. However, Gen3 variants can perform Phase 2 and Phase 3 equalization if instructed by a third-party BFM.

In the following table, signals that include lane number 0 also exist for lanes 1-7.

Table 5-9: PIPE Interface Signals

Signal	Direction	Description
`txdata0[31:0]`	Output	Transmit data <*n*>. This bus transmits data on lane <*n*>.
`txdatak0[3:0]`	Output	Transmit data control <*n*>. This signal serves as the control bit for `txdata` <*n*>. Bit 0 corresponds to the lowest-order byte of `txdata`, and so on. A value of 0 indicates a data byte. A value of 1 indicates a control byte. For Gen1 and Gen2 only.
`txblkst0`	Output	For Gen3 operation, indicates the start of a block in the transmit direction.
`txdataskip0`	Output	For Gen3 operation. Allows the MAC to instruct the TX interface to ignore the TX data interface for one clock cycle. The following encodings are defined: • 1'b0: TX data is invalid • 1'b1: TX data is valid

Signal	Direction	Description
tx_deemph0	Output	Transmit de-emphasis selection. The Arria 10 Hard IP for PCI Express sets the value for this signal based on the indication received from the other end of the link during the Training Sequences (TS). You do not need to change this value.
rxdata0[31:0] (2)	Input	Receive data <*n*>. This bus receives data on lane <*n*>.
rxdatak[3:0] (2)	Input	Receive data >*n*>. This bus receives data on lane <*n*>. Bit 0 corresponds to the lowest-order byte of rxdata, and so on. A value of 0 indicates a data byte. A value of 1 indicates a control byte. For Gen1 and Gen2 only.
rxblkst0	Input	For Gen3 operation, indicates the start of a block in the receive direction.
txdetectrx0	Output	Transmit detect receive <*n*>. This signal tells the PHY layer to start a receive detection operation or to begin loopback.
txelecidle	Output	Transmit electrical idle <*n*>. This signal forces the TX output to electrical idle.
txcompl0	Output	Transmit compliance <*n*>. This signal forces the running disparity to negative in Compliance Mode (negative COM character).
rxpolarity0	Output	Receive polarity <*n*>. This signal instructs the PHY layer to invert the polarity of the 8B/10B receiver decoding block.
powerdown0[1:0]	Output	Power down <*n*>. This signal requests the PHY to change its power state to the specified state (P0, P0s, P1, or P2).
currentcoeff0[17:0]	Output	For Gen3, specifies the coefficients to be used by the transmitter. The 18 bits specify the following coefficients: • [5:0]: C_{-1} • [11:6]: C_0 • [17:12]: C_{+1}
currentrxpreset0[2:0]	Output	For Gen3 designs, specifies the current preset.
tx_margin[2:0]	Output	Transmit V_{OD} margin selection. The value for this signal is based on the value from the Link Control 2 Register. Available for simulation only.

Signal	Direction	Description
txswing	Output	When asserted, indicates full swing for the transmitter voltage. When deasserted indicates half swing.
txsynchd0[1:0]	Output	For Gen3 operation, specifies the transmit block type. The following encodings are defined: • 2'b01: Ordered Set Block • 2'b10: Data Block
rxsynchd0[1:0]	Input	For Gen3 operation, specifies the receive block type. The following encodings are defined: • 2'b01: Ordered Set Block • 2'b10: Data Block
rxvalid0 (1)	Input	Receive valid <*n*>. This signal indicates symbol lock and valid data on rxdata<*n*> and rxdatak <*n*>.
phystatus0 (1)	Input	PHY status <*n*>. This signal communicates completion of several PHY requests.
rxelecidle0 (1)	Input	Receive electrical idle <*n*>. When asserted, indicates detection of an electrical idle.
rxstatus0[2:0] (1)	Input	Receive status <*n*>. This signal encodes receive status, including error codes for the receive data stream and receiver detection.
simu_mode_pipe	Input	When set to 1, the PIPE interface is in simulation mode.
sim_pipe_rate[1:0]	Output	The 2-bit encodings have the following meanings: • 2'b00: Gen1 rate (2.5 Gbps) • 2'b01: Gen2 rate (5.0 Gbps) • 2'b1X: Gen3 rate (8.0 Gbps)
sim_pipe_pclk_in	Input	This clock is used for PIPE simulation only, and is derived from the refclk. It is the PIPE interface clock used for PIPE mode simulation.
sim_pipe_pclk_out	Output	TX datapath clock to the BFM PHY. pclk_out is derived from refclk and provides the source synchronous clock for TX data from the PHY.
sim_pipe_clk250_out	Output	Used to generate pclk.
sim_pipe_clk500_out	Output	Used to generate pclk.

Signal	Direction	Description
sim_pipe_ltssmstate0[4:0]	Input and Output	LTSSM state: The LTSSM state machine encoding defines the following states: • 5'b00000: Detect.Quiet • 5'b 00001: Detect.Active • 5'b00010: Polling.Active • 5'b 00011: Polling.Compliance • 5'b 00100: Polling.Configuration • 5'b00101: Polling.Speed • 5'b00110: config.LinkwidthsStart • 5'b 00111: Config.Linkaccept • 5'b 01000: Config.Lanenumaccept • 5'b01001: Config.Lanenumwait • 5'b01010: Config.Complete • 5'b 01011: Config.Idle • 5'b01100: Recovery.Rcvlock • 5'b01101: Recovery.Rcvconfig • 5'b01110: Recovery.Idle • 5'b 01111: L0 • 5'b10000: Disable • 5'b10001: Loopback.Entry • 5'b10010: Loopback.Active • 5'b10011: Loopback.Exit • 5'b10100: Hot.Reset • 5'b10101: L0s • 5'b11001: L2.transmit.Wake • 5'b11010: Speed.Recovery • 5'b11011: Recovery.Equalization, Phase 0 • 5'b11100: Recovery.Equalization, Phase 1 • 5'b11101: Recovery.Equalization, Phase 2 • 5'b11110: Recovery.Equalization, Phase 3 • 5'b11111: Recovery.Equalization, Done
rxfreqlocked0 (1)	Input	When asserted indicates that the pclk_in used for PIPE simulation is valid.
rxdataskip0	Output	For Gen3 operation. Allows the PCS to instruct the RX interface to ignore the RX data interface for one clock cycle. The following encodings are defined: • 1'b0: RX data is invalid • 1'b1: RX data is valid

Signal	Direction	Description
eidleinfersel0[2:0]	Output	Electrical idle entry inference mechanism selection. The following encodings are defined: • 3'b0xx: Electrical Idle Inference not required in current LTSSM state • 3'b100: Absence of COM/SKP Ordered Set in the 128 us window for Gen1 or Gen2 • 3'b101: Absence of TS1/TS2 Ordered Set in a 1280 UI interval for Gen1 or Gen2 • 3'b110: Absence of Electrical Idle Exit in 2000 UI interval for Gen1 and 16000 UI interval for Gen2 • 3'b111: Absence of Electrical idle exit in 128 us window for Gen1

Notes:

1. These signals are for simulation only. For Quartus II software compilation, these pipe signals can be left floating.

Test Signals

Table 5-10: Test Interface Signals

The `test_in` bus provides run-time control and monitoring of the internal state of the IP core.

Signal	Direction	Description
`test_in[31:0]`	Input	The bits of the `test_in` bus have the following definitions: • [0]: Simulation mode. This signal can be set to 1 to accelerate initialization by reducing the value of many initialization counters. • [1]: Reserved. Must be set to 1'b0. • [2]: Descramble mode disable. This signal must be set to 1 during initialization in order to disable data scrambling. You can use this bit in simulation for both Endpoints and Root Ports to observe descrambled data on the link. Descrambled data cannot be used in open systems because the link partner typically scrambles the data. • [4:3]: Reserved. Must be set to 4'b01. • [5]: Compliance test mode. Disable/force compliance mode. When set, prevents the LTSSM from entering compliance mode. Toggling this bit controls the entry and exit from the compliance state, enabling the transmission of Gen1, Gen2 and Gen3 compliance patterns. • [6]: Forces entry to compliance mode when a timeout is reached in the polling.active state and not all lanes have detected their exit condition. • [7]: Disable low power state negotiation. Altera recommends setting thist bit. • [31:8]: Reserved. Set to all 0s.
`simu_mode_pipe`	Input	When asserted, simulation operates in parallel mode. When deasserted the simulation is serial.

Registers 6

2015.06.05

UG-01145_avmm

Correspondence between Configuration Space Registers and the PCIe Specification

Table 6-1: Correspondence between Configuration Space Capability Structures and PCIe Base Specification Description

For the Type 0 and Type 1 Configuration Space Headers, the first line of each entry lists Type 0 values and the second line lists Type 1 values when the values differ.

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x000:0x03C	PCI Header Type 0 Configuration Registers	Type 0 Configuration Space Header
0x000:0x03C	PCI Header Type 1 Configuration Registers	Type 1 Configuration Space Header
0x040:0x04C	Reserved	N/A
0x050:0x05C	MSI Capability Structure	MSI Capability Structure
0x068:0x070	MSI-X Capability Structure	MSI-X Capability Structure
0x070:0x074	Reserved	N/A
0x078:0x07C	Power Management Capability Structure	PCI Power Management Capability Structure
0x080:0x0B8	PCI Express Capability Structure	PCI Express Capability Structure
0x0B8:0x0FC	Reserved	N/A
0x094:0x0FF	Root Port	N/A
0x100:0x16C	Virtual Channel Capability Structure (Reserved)	Virtual Channel Capability

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x170:0x17C	Reserved	N/A
0x180:0x1FC	Virtual channel arbitration table (Reserved)	VC Arbitration Table
0x200:0x23C	Port VC0 arbitration table (Reserved)	Port Arbitration Table
0x240:0x27C	Port VC1 arbitration table (Reserved)	Port Arbitration Table
0x280:0x2BC	Port VC2 arbitration table (Reserved)	Port Arbitration Table
0x2C0:0x2FC	Port VC3 arbitration table (Reserved)	Port Arbitration Table
0x300:0x33C	Port VC4 arbitration table (Reserved)	Port Arbitration Table
0x340:0x37C	Port VC5 arbitration table (Reserved)	Port Arbitration Table
0x380:0x3BC	Port VC6 arbitration table (Reserved)	Port Arbitration Table
0x3C0:0x3FC	Port VC7 arbitration table (Reserved)	Port Arbitration Table
0x400:0x7FC	Reserved	PCIe spec corresponding section name
0x800:0x834	Advanced Error Reporting AER (optional)	Advanced Error Reporting Capability
0x838:0xFFF	Reserved	N/A
0x000	Device ID, Vendor ID	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x004	Status, Command	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x008	Class Code, Revision ID	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x00C	BIST, Header Type, Primary Latency Timer, Cache Line Size	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x010	Base Address 0	Base Address Registers
0x014	Base Address 1	Base Address Registers

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x018	Base Address 2 Secondary Latency Timer, Subordinate Bus Number, Secondary Bus Number, Primary Bus Number	Base Address Registers Secondary Latency Timer, Type 1 Configuration Space Header, Primary Bus Number
0x01C	Base Address 3 Secondary Status, I/O Limit, I/O Base	Base Address Registers Secondary Status Register ,Type 1 Configuration Space Header
0x020	Base Address 4 Memory Limit, Memory Base	Base Address Registers Type 1 Configuration Space Header
0x024	Base Address 5 Prefetchable Memory Limit, Prefetchable Memory Base	Base Address Registers Prefetchable Memory Limit, Prefetchable Memory Base
0x028	Reserved Prefetchable Base Upper 32 Bits	N/A Type 1 Configuration Space Header
0x02C	Subsystem ID, Subsystem Vendor ID Prefetchable Limit Upper 32 Bits	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x030	I/O Limit Upper 16 Bits, I/O Base Upper 16 Bits	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x034	Reserved, Capabilities PTR	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x038	Reserved	N/A
0x03C	Interrupt Pin, Interrupt Line Bridge Control, Interrupt Pin, Interrupt Line	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x050	MSI-Message Control Next Cap Ptr Capability ID	MSI and MSI-X Capability Structures
0x054	Message Address	MSI and MSI-X Capability Structures
0x058	Message Upper Address	MSI and MSI-X Capability Structures
0x05C	Reserved Message Data	MSI and MSI-X Capability Structures

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x068	MSI-X Message Control Next Cap Ptr Capability ID	MSI and MSI-X Capability Structures
0x06C	MSI-X Table Offset BIR	MSI and MSI-X Capability Structures
0x070	Pending Bit Array (PBA) Offset BIR	MSI and MSI-X Capability Structures
0x078	Capabilities Register Next Cap PTR Cap ID	PCI Power Management Capability Structure
0x07C	Data PM Control/Status Bridge Extensions Power Management Status & Control	PCI Power Management Capability Structure
0x800	PCI Express Enhanced Capability Header	Advanced Error Reporting Enhanced Capability Header
0x804	Uncorrectable Error Status Register	Uncorrectable Error Status Register
0x808	Uncorrectable Error Mask Register	Uncorrectable Error Mask Register
0x80C	Uncorrectable Error Severity Register	Uncorrectable Error Severity Register
0x810	Correctable Error Status Register	Correctable Error Status Register
0x814	Correctable Error Mask Register	Correctable Error Mask Register
0x818	Advanced Error Capabilities and Control Register	Advanced Error Capabilities and Control Register
0x81C	Header Log Register	Header Log Register
0x82C	Root Error Command	Root Error Command Register
0x830	Root Error Status	Root Error Status Register
0x834	Error Source Identification Register Correctable Error Source ID Register	Error Source Identification Register

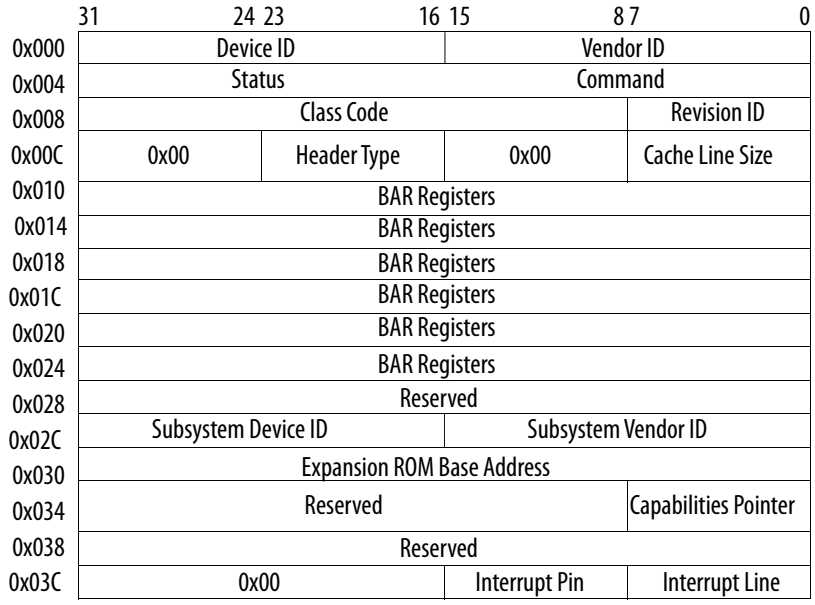
Related Information

PCI Express Base Specification 3.0

Type 0 Configuration Space Registers

Figure 6-1: Type 0 Configuration Space Registers - Byte Address Offsets and Layout

Endpoints store configuration data in the Type 0 Configuration Space. The **Correspondence between Configuration Space Registers and the PCIe Specification** on page 6-1 lists the appropriate section of the *PCI Express Base Specification* that describes these registers.



Type 1 Configuration Space Registers

Figure 6-2: Type 1 Configuration Space Registers (Root Ports)

	31 24	23 16	15 8	7 0
0x0000	Device ID		Vendor ID	
0x004	Status		Command	
0x008	Class Code			Revision ID
0x00C	BIST	Header Type	Primary Latency Timer	Cache Line Size
0x010	BAR Registers			
0x014	BAR Registers			
0x018	Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number
0x01C	Secondary Status		I/O Limit	I/O Base
0x020	Memory Limit		Memory Base	
0x024	Prefetchable Memory Limit		Prefetchable Memory Base	
0x028	Prefetchable Base Upper 32 Bits			
0x02C	Prefetchable Limit Upper 32 Bits			
0x030	I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits	
0x034	Reserved			Capabilities Pointer
0x038	Expansion ROM Base Address			
0x03C	Bridge Control		Interrupt Pin	Interrupt Line

PCI Express Capability Structures

Figure 6-3: MSI Capability Structure

	31 24	23 16	15 8	7 0
0x050	Message Control Configuration MSI Control Status Register Field Descriptions		Next Cap Ptr	Capability ID
0x054	Message Address			
0x058	Message Upper Address			
0x05C	Reserved		Message Data	

Figure 6-4: MSI-X Capability Structure

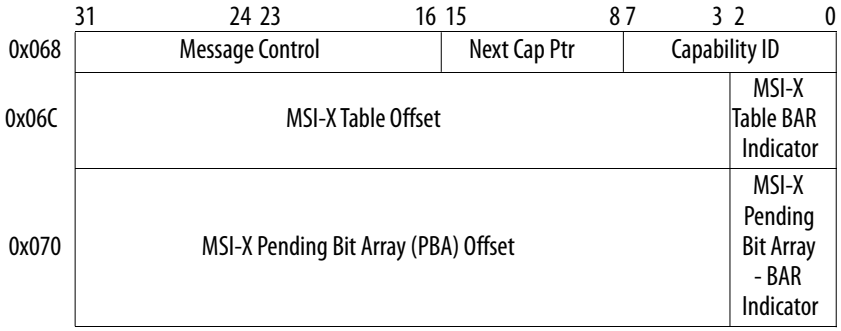


Figure 6-5: Power Management Capability Structure - Byte Address Offsets and Layout

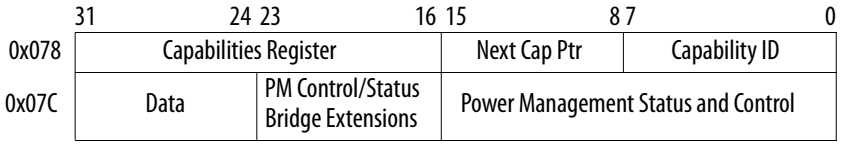


Figure 6-6: PCI Express AER Extended Capability Structure

Byte Offset	31:24	23:16	15:8	7:0
0x800	PCI Express Enhanced Capability Register			
0x804	Uncorrectable Error Status Register			
0x808	Uncorrectable Error Mask Register			
0x80C	Uncorrectable Error Severity Register			
0x810	Correctable Error Status Register			
0x814	Correctable Error Mask Register			
0x818	Advanced Error Capabilities and Control Register			
0x81C	Header Log Register			
0x82C	Root Error Command Register			
0x830	Root Error Status Register			
0x834	Error Source Identification Register		Correctable Error Source Identification Register	

Figure 6-7: PCI Express Capability Structure - Byte Address Offsets and Layout

In the following table showing the PCI Express Capability Structure, registers that are not applicable to a device are reserved.

	31 24	23 16	15 8	7 0
0x080	PCI Express Capabilities Register		Next Cap Pointer	PCI Express Capabilities ID
0x084	Device Capabilities			
0x088	Device Status		Device Control	
0x08C	Link Capabilities			
0x090	Link Status		Link Control	
0x094	Slot Capabilities			
0x098	Slot Status		Slot Control	
0x09C	Root Capabilities		Root Control	
0x0A0	Root Status			
0x0A4	Device Compatibilities 2			
0x0A8	Device Status 2		Device Control 2	
0x0AC	Link Capabilities 2			
0x0B0	Link Status 2		Link Control 2	
0x0B4	Slot Capabilities 2			
0x0B8	Slot Status 2		Slot Control 2	

Altera-Defined VSEC Registers

Figure 6-8: VSEC Registers

This extended capability structure supports Configuration via Protocol (CvP) programming and detailed internal error reporting.

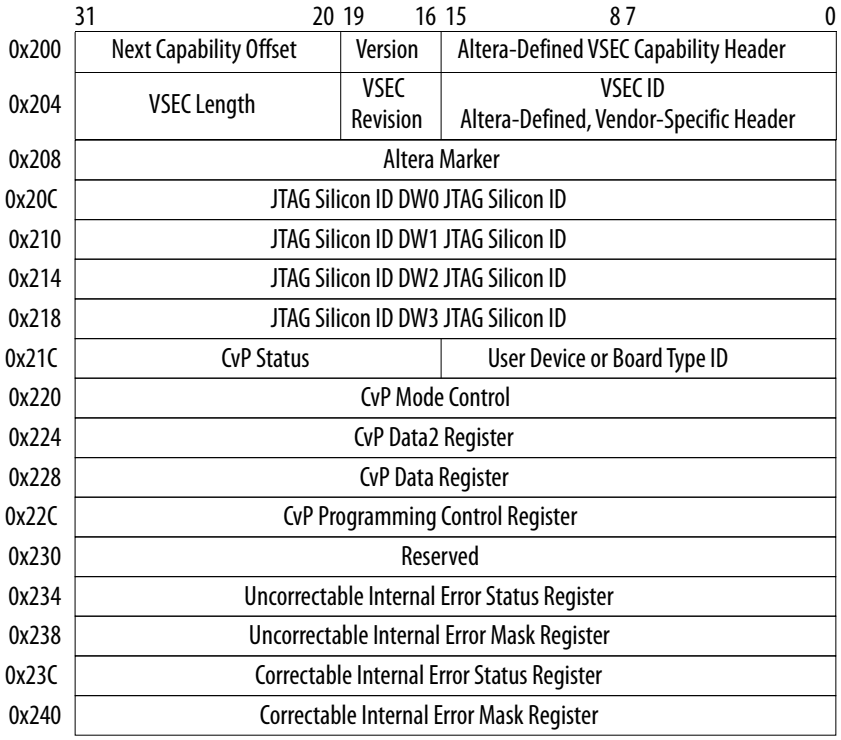


Table 6-2: Altera-Defined VSEC Capability Register, 0x200

The Altera-Defined Vendor Specific Extended Capability. This extended capability structure supports Configuration via Protocol (CvP) programming and detailed internal error reporting.

Bits	Register Description	Value	Access
[15:0]	PCI Express Extended Capability ID. Altera-defined value for VSEC Capability ID.	0x000B	RO
[19:16]	Version. Altera-defined value for VSEC version.	0x1	RO
[31:20]	Next Capability Offset. Starting address of the next Capability Structure implemented, if any.	Variable	RO

Table 6-3: Altera-Defined Vendor Specific Header

You can specify these values when you instantiate the Hard IP. These registers are read-only at run-time.

Bits	Register Description	Value	Access
[15:0]	`VSEC ID`. A user configurable VSEC ID.	User entered	RO
[19:16]	`VSEC Revision`. A user configurable VSEC revision.	Variable	RO
[31:20]	`VSEC Length`. Total length of this structure in bytes.	0x044	RO

Table 6-4: Altera Marker Register

Bits	Register Description	Value	Access
[31:0]	`Altera Marker`. This read only register is an additional marker. If you use the standard Altera Programmer software to configure the device with CvP, this marker provides a value that the programming software reads to ensure that it is operating with the correct VSEC.	A Device Value	RO

Table 6-5: JTAG Silicon ID Register

Bits	Register Description	Value	Access
[127:96]	`JTAG Silicon ID DW3`	Application Specific	RO
[95:64]	`JTAG Silicon ID DW2`	Application Specific	RO
[63:32]	`JTAG Silicon ID DW1`	Application Specific	RO
[31:0]	`JTAG Silicon ID DW0`. This is the JTAG Silicon ID that CvP programming software reads to determine that the correct SRAM object file (**.sof**) is being used.	Application Specific	RO

Table 6-6: User Device or Board Type ID Register

Bits	Register Description	Value	Access
[15:0]	Configurable device or board type ID to specify to CvP the correct **.sof**.	Variable	RO

CvP Registers

Table 6-7: CvP Status

The `CvP Status` register allows software to monitor the CvP status signals.

Bits	Register Description	Reset Value	Access
[31:26]	Reserved	0x00	RO
[25]	`PLD_CORE_READY`. From FPGA fabric. This status bit is provided for debug.	Variable	RO
[24]	`PLD_CLK_IN_USE`. From clock switch module to fabric. This status bit is provided for debug.	Variable	RO
[23]	`CVP_CONFIG_DONE`. Indicates that the FPGA control block has completed the device configuration via CvP and there were no errors.	Variable	RO
[22]	Reserved	Variable	RO
[21]	`USERMODE`. Indicates if the configurable FPGA fabric is in user mode.	Variable	RO
[20]	`CVP_EN`. Indicates if the FPGA control block has enabled CvP mode.	Variable	RO
[19]	`CVP_CONFIG_ERROR`. Reflects the value of this signal from the FPGA control block, checked by software to determine if there was an error during configuration.	Variable	RO
[18]	`CVP_CONFIG_READY`. Reflects the value of this signal from the FPGA control block, checked by software during programming algorithm.	Variable	RO
[17:0]	Reserved	Variable	RO

Table 6-8: CvP Mode Control

The `CvP Mode Control` register provides global control of the CvP operation.

Bits	Register Description	Reset Value	Access
[31:16]	Reserved.	0x0000	RO
[15:8]	`CVP_NUMCLKS`. This is the number of clocks to send for every CvP data write. Set this field to one of the values below depending on your configuration image: • 0x01 for uncompressed and unencrypted images • 0x04 for uncompressed and encrypted images • 0x08 for all compressed images	0x00	RW
[7:3]	Reserved.	0x0	RO
[2]	`CVP_FULLCONFIG`. Request that the FPGA control block reconfigure the entire FPGA including the Arria 10 Hard IP for PCI Express, bring the PCIe link down.	1'b0	RW

Bits	Register Description	Reset Value	Access
[1]	HIP_CLK_SEL. Selects between PMA and fabric clock when USER_MODE = 1 and PLD_CORE_READY = 1. The following encodings are defined: • 1: Selects internal clock from PMA which is required for CVP_MODE. • 0: Selects the clock from soft logic fabric. This setting should only be used when the fabric is configured in USER_MODE with a configuration file that connects the correct clock. To ensure that there is no clock switching during CvP, you should only change this value when the Hard IP for PCI Express has been idle for 10 µs and wait 10 µs after changing this value before resuming activity.	1'b0	RW
[0]	CVP_MODE. Controls whether the IP core is in CVP_MODE or normal mode. The following encodings are defined: • 1:CVP_MODE is active. Signals to the FPGA control block active and all TLPs are routed to the Configuration Space. This CVP_MODE cannot be enabled if CVP_EN = 0. • 0: The IP core is in normal mode and TLPs are routed to the FPGA fabric.	1'b0	RW

Table 6-9: CvP Data Registers

The following table defines the CvP Data registers. For 64-bit data, the optional CvP Data2 stores the upper 32 bits of data. Programming software should write the configuration data to these registers. If you Every write to these register sets the data output to the FPGA control block and generates <*n*> clock cycles to the FPGA control block as specified by the CVP_NUM_CLKS field in the CvP Mode Control register. Software must ensure that all bytes in the memory write dword are enabled. You can access this register using configuration writes, alternatively, when in CvP mode, these registers can also be written by a memory write to any address defined by a memory space BAR for this device. Using memory writes should allow for higher throughput than configuration writes.

Bits	Register Description	Reset Value	Access
[31:0]	Upper 32 bits of configuration data to be transferred to the FPGA control block to configure the device. You can choose 32- or 64-bit data.	0x00000000	RW
[31:0]	Lower 32 bits of configuration data to be transferred to the FPGA control block to configure the device.	0x00000000	RW

Table 6-10: CvP Programming Control Register

This register is written by the programming software to control CvP programming.

Bits	Register Description	Reset Value	Access
[31:2]	Reserved.	0x0000	RO

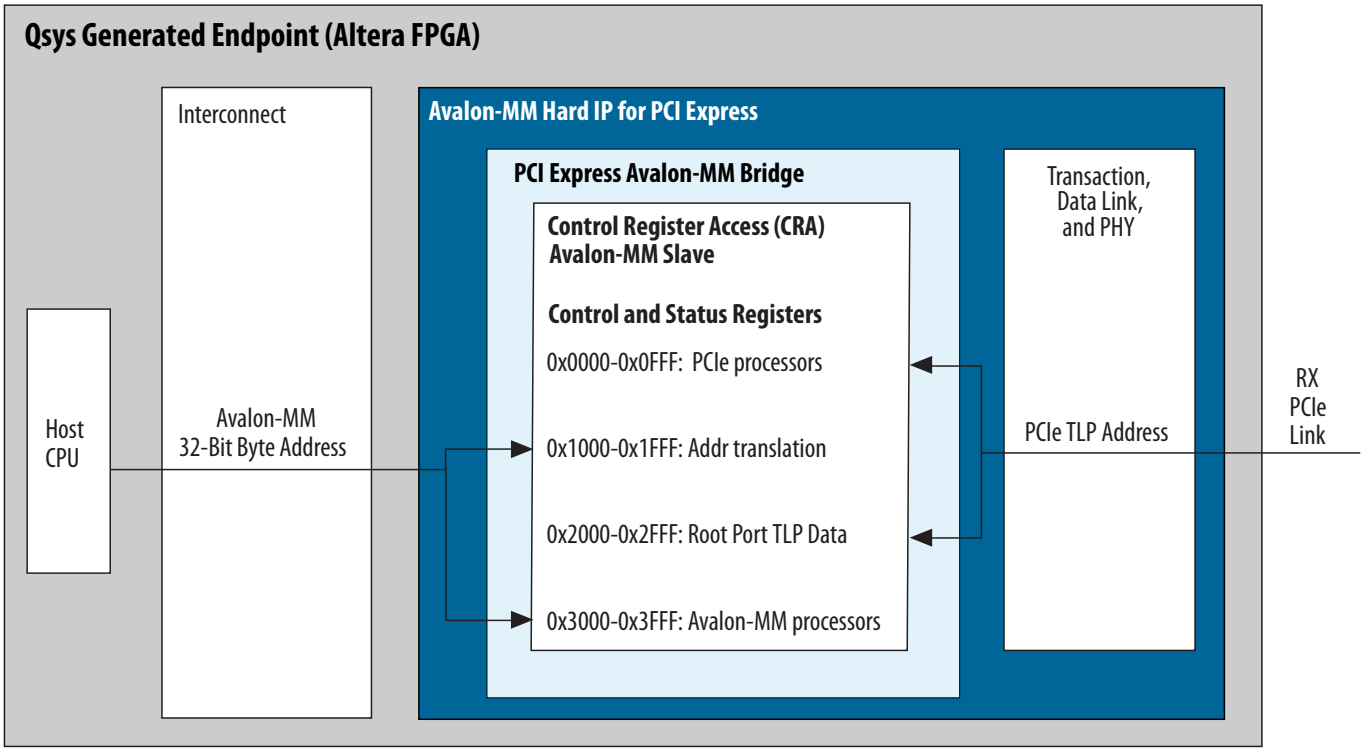
Bits	Register Description	Reset Value	Access
[1]	START_XFER. Sets the CvP output to the FPGA control block indicating the start of a transfer.	1'b0	RW
[0]	CVP_CONFIG. When asserted, instructs that the FPGA control block begin a transfer via CvP.	1'b0	RW

64- or 128-Bit Avalon-MM Bridge Register Descriptions

The CRA Avalon-MM slave module provides access control and status registers in the PCI Express Avalon-MM bridge. In addition, it provides access to selected Configuration Space registers and link status registers in read-only mode. This module is optional. However, you must include it to access the registers.

The control and status register address space is 16 KBytes. Each 4-KByte sub-region contains a set of functions, which may be specific to accesses from the PCI Express Root Complex only, from Avalon-MM processors only, or from both types of processors. Because all accesses come across the interconnect fabric —requests from the Avalon-MM Arria 10 Hard IP for PCI Express are routed through the interconnect fabric—hardware does not enforce restrictions to limit individual processor access to specific regions. However, the regions are designed to enable straight-forward enforcement by processor software. The following figure illustrates accesses to the Avalon-MM control and status registers from the Host CPU and PCI Express link.

Figure 6-9: Accesses to the Avalon-MM Bridge Control and Status Register



The following table describes the four subregions.

Table 6-11: Avalon-MM Control and Status Register Address Spaces

AddressRange	Address Space Usage
0x0000-0x0FFF	Registers typically intended for access by PCI Express link partner only. This includes PCI Express interrupt enable controls, write access to the PCI Express Avalon-MM bridge mailbox registers, and read access to Avalon-MM-to-PCI Express mailbox registers.
0x1000-0x1FFF	Avalon-MM-to-PCI Express address translation tables. Depending on the system design these may be accessed by the PCI Express link partner, Avalon-MM processors, or both.
0x2000-0x2FFF	Root Port request registers. An embedded processor, such as the Nios II processor, programs these registers to send the data for Configuration TLPs, I/O TLPs, single dword Memory Read and Write requests, and receive interrupts from an Endpoint.
0x3000-0x3FFF	Registers typically intended for access by Avalon-MM processors only. Provides host access to selected Configuration Space and status registers.

Note: The data returned for a read issued to any undefined address in this range is unpredictable.

The following table lists the complete address map for the PCI Express Avalon-MM bridge registers.

Note: In the following table the text in green are links to the detailed register description

Table 6-12: PCI Express Avalon-MM Bridge Register Map

Address Range	Register
0x0040	Avalon-MM to PCI Express Interrupt Status Register
0x0050	Avalon-MM to PCI Express Interrupt Status Enable Register
0x0800–0x081F	PCI Express-to-Avalon-MM Mailbox Registers
0x0900–x091F	Avalon-MM to PCI Express Mailbox Registers
0x1000–0x1FFF	Avalon-MM to PCI Express Address Translation Table
0x2000–0x2FFF	Root Port TLP Data Registers
0x3060	Avalon-MM to PCI Express Interrupt Status Registers for Root Ports
0x3060	PCI Express to Avalon-MM Interrupt Status Register for Endpoints

Address Range	Register
0x3070	INT-X Interrupt Enable Register for Root Ports
0x3070	INT-X Interrupt Enable Register for Endpoints
0x3A00-0x3A1F	Avalon-MM to PCI Express Mailbox Registers
0x3B00-0x3B1F	PCI Express to Avalon-MM Mailbox Registers
0x3C00-0x3C6C	Host (Avalon-MM master) access to selected Configuration Space and status registers.

Avalon-MM to PCI Express Interrupt Registers

Avalon-MM to PCI Express Interrupt Status Registers

These registers contain the status of various signals in the PCI Express Avalon-MM bridge logic and allow PCI Express interrupts to be asserted when enabled. Only Root Complexes should access these registers; however, hardware does not prevent other Avalon-MM masters from accessing them.

Table 6-13: Avalon-MM to PCI Express Interrupt Status Register, 0x0040

Bit	Name	Access	Description
[31:24]	Reserved	N/A	N/A
[23]	A2P_MAILBOX_INT7	RW1C	1 when the A2P_MAILBOX7 is written to
[22]	A2P_MAILBOX_INT6	RW1C	1 when the A2P_MAILBOX6 is written to
[21]	A2P_MAILBOX_INT5	RW1C	1 when the A2P_MAILBOX5 is written to
[20]	A2P_MAILBOX_INT4	RW1C	1 when the A2P_MAILBOX4 is written to
[19]	A2P_MAILBOX_INT3	RW1C	1 when the A2P_MAILBOX3 is written to
[18]	A2P_MAILBOX_INT2	RW1C	1 when the A2P_MAILBOX2 is written to
[17]	A2P_MAILBOX_INT1	RW1C	1 when the A2P_MAILBOX1 is written to
[16]	A2P_MAILBOX_INT0	RW1C	1 when the A2P_MAILBOX0 is written to

Bit	Name	Access	Description
[15:0]	AVL_IRQ_ASSERTED[15:0]	RO	Current value of the Avalon-MM interrupt (IRQ) input ports to the Avalon-MM RX master port: • 0—Avalon-MM IRQ is not being signaled. • 1—Avalon-MM IRQ is being signaled. A Qsys-generated IP Compiler for PCI Express has as many as 16 distinct IRQ input ports. Each AVL_IRQ_ASSERTED[] bit reflects the value on the corresponding IRQ input port.

Avalon-MM to PCI Express Interrupt Enable Registers

A PCI Express interrupt can be asserted for any of the conditions registered in the Avalon-MM to PCI Express Interrupt Status register by setting the corresponding bits in the Avalon-MM-to-PCI Express Interrupt Enable register. Either MSI or legacy interrupts can be generated as explained in the section *Enabling MSI or Legacy Interrupts*

Table 6-14: Avalon-MM to PCI Express Interrupt Enable Register, 0x0050

Bits	Name	Access	Description
[31:24]	Reserved	N/A	N/A
[23:16]	A2P_MB_IRQ	RW	Enables generation of PCI Express interrupts when a specified mailbox is written to by an external Avalon-MM master.
[4:0]	AVL_IRQ[15:0]	RW	Enables generation of PCI Express interrupts when a specified Avalon-MM interrupt signal is asserted. Your Qsys system may have as many as 16 individual input interrupt signals.

Table 6-15: Avalon-MM Interrupt Vector Register - 0x0060

Bits	Name	Access	Description
[31:5]	Reserved	N/A	N/A

Bits	Name	Access	Description
[15:0]	AVL_IRQ_Vector	RO	Stores the interrupt vector of the system interconnect fabric. The host software should read this register after being interrupted and determine the servicing priority.

PCI Express Mailbox Registers

The PCI Express Root Complex typically requires write access to a set of PCI Express-to-Avalon-MM mailbox registers and read-only access to a set of Avalon-MM-to-PCI Express mailbox registers. Eight mailbox registers are available.

The PCI Express-to-Avalon-MM Mailbox registers are writable at the addresses shown in the following table. Writing to one of these registers causes the corresponding bit in the Avalon-MM Interrupt Status register to be set to a one.

Table 6-16: PCI Express-to-Avalon-MM Mailbox Registers, 0x0800–0x081F

Address	Name	Access	Description
0x0800	P2A_MAILBOX0	RW	PCI Express-to-Avalon-MM Mailbox 0
0x0804	P2A_MAILBOX1	RW	PCI Express-to-Avalon-MM Mailbox 1
0x0808	P2A_MAILBOX2	RW	PCI Express-to-Avalon-MM Mailbox 2
0x080C	P2A_MAILBOX3	RW	PCI Express-to-Avalon-MM Mailbox 3
0x0810	P2A_MAILBOX4	RW	PCI Express-to-Avalon-MM Mailbox 4
0x0814	P2A_MAILBOX5	RW	PCI Express-to-Avalon-MM Mailbox 5
0x0818	P2A_MAILBOX6	RW	PCI Express-to-Avalon-MM Mailbox 6
0x081C	P2A_MAILBOX7	RW	PCI Express-to-Avalon-MM Mailbox 7

The Avalon-MM-to-PCI Express Mailbox registers are read at the addresses shown in the following table. The PCI Express Root Complex should use these addresses to read the mailbox information after being signaled by the corresponding bits in the Avalon MM to PCI Express Interrupt Status register.

Table 6-17: Avalon-MM-to-PCI Express Mailbox Registers, 0x0900–0x091F

Address	Name	Access	Description
0x0900	A2P_MAILBOX0	RO	Avalon-MM-to-PCI Express Mailbox 0

Address	Name	Access	Description
0x0904	A2P_MAILBOX1	RO	Avalon-MM-to-PCI Express Mailbox 1
0x0908	A2P_MAILBOX2	RO	Avalon-MM-to-PCI Express Mailbox 2
0x090C	A2P_MAILBOX3	RO	Avalon-MM-to-PCI Express Mailbox 3
0x0910	A2P_MAILBOX4	RO	Avalon-MM-to-PCI Express Mailbox 4
0x0914	A2P_MAILBOX5	RO	Avalon-MM-to-PCI Express Mailbox 5
0x0918	A2P_MAILBOX6	RO	Avalon-MM-to-PCI Express Mailbox 6
0x091C	A2P_MAILBOX7	RO	Avalon-MM-to-PCI Express Mailbox 7

Avalon-MM-to-PCI Express Address Translation Table

The Avalon-MM-to-PCI Express address translation table is writable using the CRA slave port. Each entry in the PCI Express address translation table is 8 bytes wide, regardless of the value in the current PCI Express address width parameter. Therefore, register addresses are always the same width, regardless of PCI Express address width.

These table entries are repeated for each address specified in the **Number of address pages** parameter. If **Number of address pages** is set to the maximum of 512, 0x1FF8 contains A2P_ADDR_SPACE511 and A2P_ADDR_MAP_LO511 and 0x1FFC contains A2P_ADDR_MAP_HI511.

Table 6-18: Avalon-MM-to-PCI Express Address Translation Table, 0x1000–0x1FFF

Address	Bits	Name	Access	Description
0x1000	[1:0]	A2P_ADDR_SPACE0	RW	Address space indication for entry 0. The following encodings are defined: • 2'b00:. Memory Space, 32-bit PCI Express address. 32-bit header is generated. Address bits 63:32 of the translation table entries are ignored. • 2'b01: Memory space, 64-bit PCI Express address. 64-bit address header is generated. • 2'b10: Reserved • 2'b11: Reserved
	[31:2]	A2P_ADDR_MAP_LO0	RW	Lower bits of Avalon-MM-to-PCI Express address map entry 0.
0x1004	[31:0]	A2P_ADDR_MAP_HI0	RW	Upper bits of Avalon-MM-to-PCI Express address map entry 0.

Address	Bits	Name	Access	Description
0x1008	[1:0]	A2P_ADDR_SPACE1	RW	Address space indication for entry 1. This entry is available only if the number of translation table entries (**Number of address pages**) is greater than 1. The same encodings are defined for A2P_ADDR_SPACE1 as for A2P_ADDR_SPACE0.:
	[31:2]	A2P_ADDR_MAP_LO1	RW	Lower bits of Avalon-MM-to-PCI Express address map entry 1. This entry is only implemented if the number of address translation table entries is greater than 1.
0x100C	[31:0]	A2P_ADDR_MAP_HI1	RW	Upper bits of Avalon-MM-to-PCI Express address map entry 1. This entry is only implemented if the number of address translation table entries is greater than 1.

PCI Express to Avalon-MM Interrupt Status and Enable Registers for Endpoints

The registers in this section contain status of various signals in the PCI Express Avalon-MM bridge logic and allow Avalon interrupts to be asserted when enabled. A processor local to the interconnect fabric that processes the Avalon-MM interrupts can access these registers.

Note: These registers must not be accessed by the PCI Express Avalon-MM bridge master ports; however, there is nothing in the hardware that prevents a PCI Express Avalon-MM bridge master port from accessing these registers.

The following table describes the Interrupt Status register when you configure the core as an Endpoint. It records the status of all conditions that can cause an Avalon-MM interrupt to be asserted.

Table 6-19: PCI Express to Avalon-MM Interrupt Status Register for Endpoints, 0x3060

Bits	Name	Access	Description
0	ERR_PCI_WRITE_FAILURE	RW1C	When set to 1, indicates a PCI Express write failure. This bit can also be cleared by writing a 1 to the same bit in the Avalon MM to PCI Express Interrupt Status register.
1	ERR_PCI_READ_FAILURE	RW1C	When set to 1, indicates the failure of a PCI Express read. This bit can also be cleared by writing a 1 to the same bit in the Avalon MM to PCI Express Interrupt Status register.

Bits	Name	Access	Description
[15:2]	Reserved	—	—
[16]	P2A_MAILBOX_INT0	RW1C	1 when the P2A_MAILBOX0 is written
[17]	P2A_MAILBOX_INT1	RW1C	1 when the P2A_MAILBOX1 is written
[18]	P2A_MAILBOX_INT2	RW1C	1 when the P2A_MAILBOX2 is written
[19]	P2A_MAILBOX_INT3	RW1C	1 when the P2A_MAILBOX3 is written
[20]	P2A_MAILBOX_INT4	RW1C	1 when the P2A_MAILBOX4 is written
[21]	P2A_MAILBOX_INT5	RW1C	1 when the P2A_MAILBOX5 is written
[22]	P2A_MAILBOX_INT6	RW1C	1 when the P2A_MAILBOX6 is written
[23]	P2A_MAILBOX_INT7	RW1C	1 when the P2A_MAILBOX7 is written
[31:24]	Reserved	—	—

An Avalon-MM interrupt can be asserted for any of the conditions noted in the `Avalon-MM Interrupt Status` register by setting the corresponding bits in the `PCI Express to Avalon-MM Interrupt Enable` register.

PCI Express interrupts can also be enabled for all of the error conditions described. However, it is likely that only one of the Avalon-MM or PCI Express interrupts can be enabled for any given bit. Typically, a single process in either the PCI Express or Avalon-MM domain handles the condition reported by the interrupt.

Table 6-20: INT-X Interrupt Enable Register for Endpoints, 0x3070

Bits	Name	Access	Description
[31:0]	PCI Express to Avalon-MM Interrupt Enable	RW	When set to 1, enables the interrupt for the corresponding bit in the PCI Express to Avalon MM Interrupt Status register to cause the Avalon Interrupt signal (cra_Irq_o) to be asserted. Only bits implemented in the PCI Express to Avalon MM Interrupt Status register are implemented in the Enable register. Reserved bits cannot be set to a 1.

Avalon-MM Mailbox Registers

A processor local to the interconnect fabric typically requires write access to a set of `Avalon-MM-to-PCI Express Mailbox` registers and read-only access to a set of `PCI Express-to-Avalon-MM Mailbox` registers. Eight mailbox registers are available.

The `Avalon-MM-to-PCI Express Mailbox` registers are writable at the addresses shown in the following table. When the Avalon-MM processor writes to one of these registers the corresponding bit in the `Avalon MM to PCI Express Interrupt Status` register is set to 1.

Table 6-21: Avalon-MM to PCI Express Mailbox Registers, 0x3A00–0x3A1F

Address	Name	Access	Description
0x3A00	A2P_MAILBOX0	RW	Avalon-MM-to-PCI Express mailbox 0
0x3A04	A2P_MAILBOX1	RW	Avalon-MM-to-PCI Express mailbox 1
0x3A08	A2P _MAILBOX2	RW	Avalon-MM-to-PCI Express mailbox 2
0x3A0C	A2P _MAILBOX3	RW	Avalon-MM-to-PCI Express mailbox 3
0x3A10	A2P _MAILBOX4	RW	Avalon-MM-to-PCI Express mailbox 4
0x3A14	A2P _MAILBOX5	RW	Avalon-MM-to-PCI Express mailbox 5
0x3A18	A2P _MAILBOX6	RW	Avalon-MM-to-PCI Express mailbox 6
0x3A1C	A2P_MAILBOX7	RW	Avalon-MM-to-PCI Express mailbox 7

The `PCI Express-to-Avalon-MM Mailbox` registers are read-only at the addresses shown in the following table. The Avalon-MM processor reads these registers when the corresponding bit in the `PCI Express to Avalon-MM Interrupt Status` register is set to 1.

Table 6-22: PCI Express to Avalon-MM Mailbox Registers, 0x3B00–0x3B1F

Address	Name	Access Mode	Description
0x3B00	P2A_MAILBOX0	RO	PCI Express-to-Avalon-MM mailbox 0
0x3B04	P2A_MAILBOX1	RO	PCI Express-to-Avalon-MM mailbox 1
0x3B08	P2A_MAILBOX2	RO	PCI Express-to-Avalon-MM mailbox 2
0x3B0C	P2A_MAILBOX3	RO	PCI Express-to-Avalon-MM mailbox 3
0x3B10	P2A_MAILBOX4	RO	PCI Express-to-Avalon-MM mailbox 4

Address	Name	Access Mode	Description
0x3B14	P2A_MAILBOX5	RO	PCI Express-to-Avalon-MM mailbox 5
0x3B18	P2A_MAILBOX6	RO	PCI Express-to-Avalon-MM mailbox 6
0x3B1C	P2A_MAILBOX7	RO	PCI Express-to-Avalon-MM mailbox 7

Control Register Access (CRA) Avalon-MM Slave Port

Table 6-23: Configuration Space Register Descriptions

For registers that are less than 32 bits, the upper bits are unused.

Byte Offset	Register	Dir	Description
14'h3C00	cfg_dev_ctrl[15:0]	O	cfg_devctrl[15:0] is device control for the PCI Express capability structure.
14'h3C04	cfg_dev_ctrl2[15:0]	O	cfg_dev2ctrl[15:0] is device control 2 for the PCI Express capability structure.
14'h3C08	cfg_link_ctrl[15:0]	O	cfg_link_ctrl[15:0]is the primary Link Control of the PCI Express capability structure. For Gen2 or Gen3 operation, you must write a 1'b1 to Retrain Link bit (Bit[5] of the cfg_link_ctrl) of the Root Port to initiate retraining to a higher data rate after the initial link training to Gen1 L0 state. Retraining directs the LTSSM to the Recovery state. Retraining to a higher data rate is not automatic for the Arria 10 Hard IP for PCI Express IP Core even if both devices on the link are capable of a higher data rate.
14'h3C0C	cfg_link_ctrl2[15:0]	O	cfg_link_ctrl2[31:16] is the secondary Link Control register of the PCI Express capability structure for Gen2 operation. When tl_cfg_addr=2, tl_cfg_ctl returns the primary and secondary Link Control registers, {cfg_link_ctrl[15:0], cfg_link_ctrl2[15:0]}, the primary Link Status register contents is available on tl_cfg_sts[46:31]. For Gen1 variants, the link bandwidth notification bit is always set to 0. For Gen2 variants, this bit is set to 1.

Byte Offset	Register	Dir	Description
14'h3C10	cfg_prm_cmd[15:0]	O	Base/Primary Command register for the PCI Configuration Space.
14'h3C14	cfg_root_ctrl[7:0]	O	Root control and status register of the PCI-Express capability. This register is only available in Root Port mode.
14'h3C18	cfg_sec_ctrl[15:0]	O	Secondary bus Control and Status register of the PCI-Express capability. This register is only available in Root Port mode.
14'h3C1C	cfg_secbus[7:0]	O	Secondary bus number. Available in Root Port mode.
14'h3C20	cfg_subbus[7:0]	O	Subordinate bus number. Available in Root Port mode.
14'h3C24	cfg_msi_addr_low[31:0]	O	cfg_msi_add[31:0] is the MSI message address.
14'h3C28	cfg_msi_addr_hi[63:32]	O	cfg_msi_add[63:32] is the MSI upper message address.
14'h3C2C	cfg_io_bas[19:0]	O	The IO base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C30	cfg_io_lim[19:0]	O	The IO limit register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C34	cfg_np_bas[11:0]	O	The non-prefetchable memory base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C38	cfg_np_lim[11:0]	O	The non-prefetchable memory limit register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C3C	cfg_pr_bas_low[31:0]	O	The lower 32 bits of the prefetchable base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C40	cfg_pr_bas_hi[43:32]	O	The upper 12 bits of the prefetchable base registers of the Type1 Configuration Space. This register is only available in Root Port mode.

Byte Offset	Register	Dir	Description
14'h3C44	cfg_pr_lim_low[31:0]	O	The lower 32 bits of the prefetchable limit registers of the Type1 Configuration Space. Available in Root Port mode.
14'h3C48	cfg_pr_lim_hi[43:32]	O	The upper 12 bits of the prefetchable limit registers of the Type1 Configuration Space. Available in Root Port mode.
14'h3C4C	cfg_pmcsr[31:0]	O	cfg_pmcsr[31:16] is Power Management Control and cfg_pmcsr[15:0]is the Power Management Status register.
14'h3C50	cfg_msixcsr[15:0]	O	MSI-X message control register.
14'h3C54	cfg_msicsr[15:0]	O	MSI message control.
14'h3C58	cfg_tcvcmap[23:0]	O	Configuration traffic class (TC)/virtual channel (VC) mapping. The Application Layer uses this signal to generate a TLP mapped to the appropriate channel based on the traffic class of the packet. The following encodings are defined: • cfg_tcvcmap[2:0]: Mapping for TC0 (always 0) . • cfg_tcvcmap[5:3]: Mapping for TC1. • cfg_tcvcmap[8:6]: Mapping for TC2. • cfg_tcvcmap[11:9]: Mapping for TC3. • cfg_tcvcmap[14:12]: Mapping for TC4. • cfg_tcvcmap[17:15]: Mapping for TC5. • cfg_tcvcmap[20:18]: Mapping for TC6. • cfg_tcvcmap[23:21]: Mapping for TC7.
14'h3C5C	cfg_msi_data[15:0]	O	cfg_msi_data[15:0] is message data for MSI.
14'h3C60	cfg_busdev[12:0]	O	Bus/Device Number captured by or programmed in the Hard IP.
14'h3C64	ltssm_reg[4:0]	O	Specifies the current LTSSM state. The LTSSM state machine encoding defines the following states: • 00000: Detect.Quiet • 00001: Detect.Active • 00010: Polling.Active • 00011: Polling.Compliance • 00100: Polling.Configuration

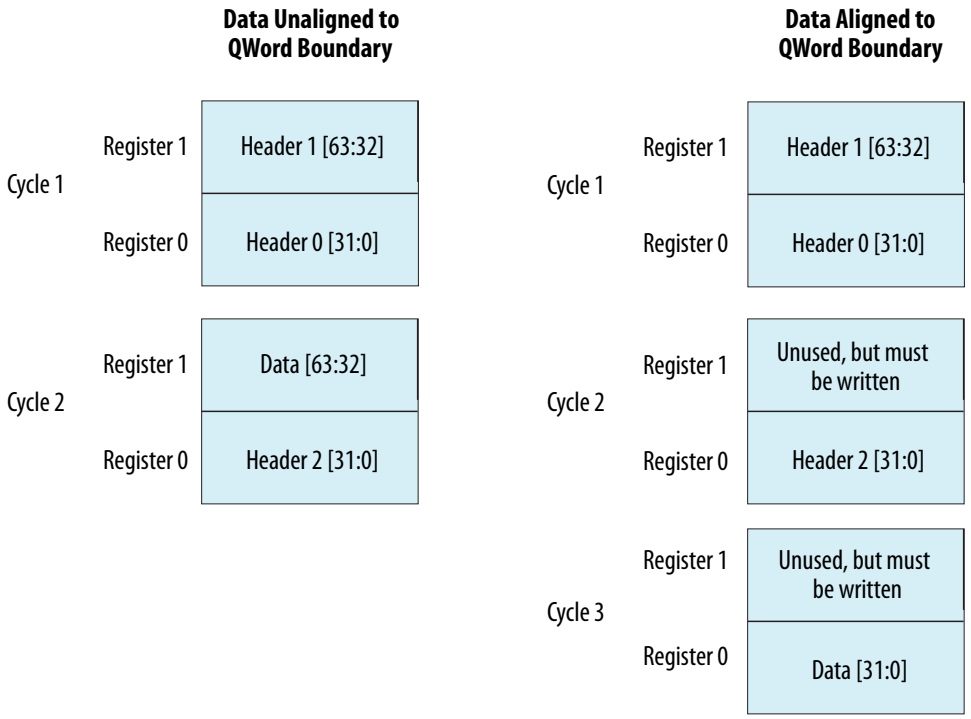
Byte Offset	Register	Dir	Description
			• 00101: Polling.Speed • 00110: config.Linkwidthstart • 00111: Config.Linkaccept • 01000: Config.Lanenumaccept • 01001: Config.Lanenumwait • 01010: Config.Complete • 01011: Config.Idle • 01100: Recovery.Rcvlock • 01101: Recovery.Rcvconfig • 01110: Recovery.Idle • 01111: L0 • 10000: Disable • 10001: Loopback.Entry • 10010: Loopback.Active • 10011: Loopback.Exit • 10100: Hot.Reset • 10101: LOs • 11001: L2.transmit.Wake • 11010: Speed.Recovery • 11011: Recovery.Equalization, Phase 0 • 11100: Recovery.Equalization, Phase 1 • 11101: Recovery.Equalization, Phase 2 • 11110: recovery.Equalization, Phase 3
14'h3C68	`current_speed_reg[1:0]`	O	Indicates the current speed of the PCIe link. The following encodings are defined: • 2b'00: Undefined • 2b'01: Gen1 • 2b'10: Gen2 • 2b'11: Gen3
14'h3C6C	`lane_act_reg[3:0]`	O	Lane Active Mode: This signal indicates the number of lanes that configured during link training. The following encodings are defined: • 4'b0001: 1 lane • 4'b0010: 2 lanes • 4'b0100: 4 lanes • 4'b1000: 8 lanes

Programming Model for Avalon-MM Root Port

The Application Layer writes the Root Port TLP TX Data registers with TLP formatted data for Configuration Read and Write Requests, Message TLPs, Message TLPs with data payload, I/O Read and Write Requests, or single dword Memory Read and Write Requests. Software should check the Root Port `Link Status` register (offset 0x92) to ensure the Data Link Layer Link `Active` bit is set to 1'b1 before issuing a Configuration request to downstream ports.

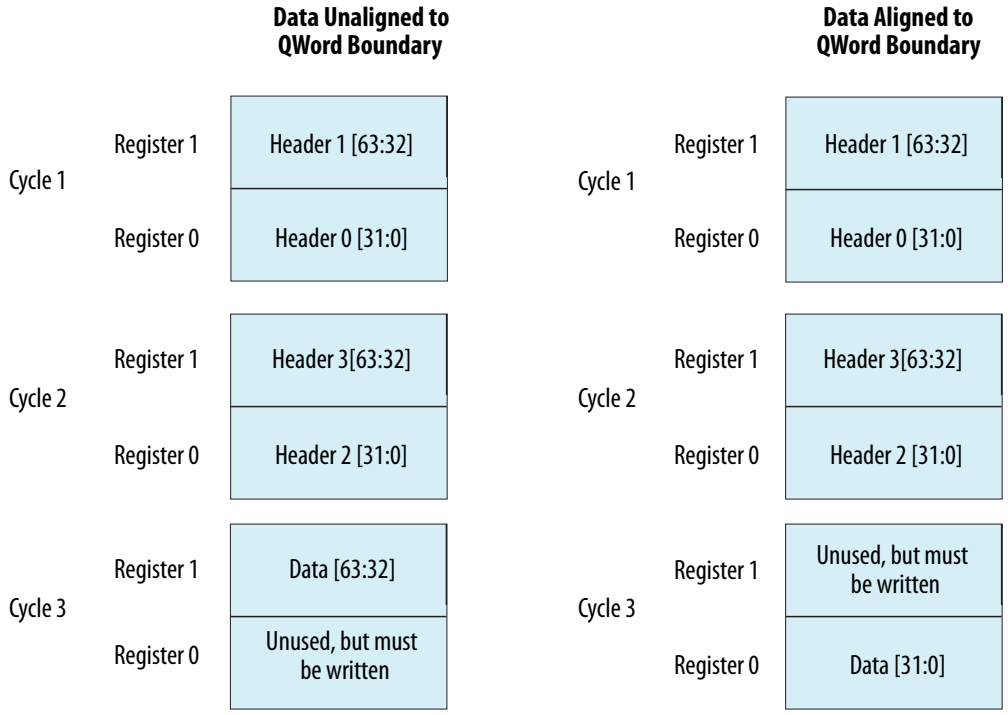
The Application Layer data must be in the appropriate TLP format with the data payload aligned to the TLP address. Aligning the payload data to the TLP address may result in the payload data being either aligned or unaligned to the qword. The following figure illustrates three dword TLPs with data that is aligned and unaligned to the qword.

Figure 6-10: Layout of Data with 3 Dword Headers



The following figure illustrates four dword TLPs with data that are aligned and unaligned to the qword.

Figure 6-11: Layout of Data with 4 Dword Headers



The TX TLP programming model scales with the data width. The Application Layer performs the same writes for both the 64- and 128-bit interfaces. The Application Layer can only have one outstanding non-posted request at a time. The Application Layer must use tags 16–31 to identify non-posted requests.

Note: For Root Ports, the Avalon-MM bridge does not filter Type 0 Configuration Requests by device number. Application Layer software should filter out all requests to Avalon-MM Root Port registers that are not for device 0. Application Layer software should return an Unsupported Request Completion Status.

Sending a Write TLP

The Application Layer performs the following sequence of Avalon-MM accesses to the CRA slave port to send a Memory Write Request:

1. Write the first 32 bits of the TX TLP to `RP_TX_REG0`.
2. Write the next 32 bits of the TX TLP to `RP_TX_REG1`.
3. Write the `RP_TX_CNTRL.SOP` to 1'b1 to push the first two dwords of the TLP into the Root Port TX FIFO.
4. Repeat Steps 1 and 2. The second write to `RP_TX_REG1` is required, even for three dword TLPs with aligned data.
5. If the packet is complete, write `RP_TX_CNTRL` to 2'b10 to indicate the end of the packet. If the packet is not complete, write 2'b00 to `RP_TX_CNTRL`.
6. Repeat this sequence to program a complete TLP.

When the programming of the TX TLP is complete, the Avalon-MM bridge schedules the TLP with higher priority than TX TLPs coming from the TX slave port.

Sending a Read TLP or Receiving a Non-Posted Completion TLP

The TLPs associated with the Non-Posted TX requests are stored in the RP_RX_CPL FIFO buffer and subsequently loaded into RP_RXCPL registers. The Application Layer performs the following sequence to retrieve the TLP.

1. Polls the `RP_RXCPL_STA TUS.SOP` to determine when it is set to 1'b1.
2. Then `RP_RXCPL_STATUS.SOP` = 1'b'1, reads `RP_RXCPL_REG0` and `RP_RXCPL_REG1` to retrieve dword 0 and dword 1 of the TLP.
3. Read the `RP_RXCPL_STATUS.EOP`.
   - If `RP_RXCPL_STATUS.EOP` = 1'b0, read `RP_RXCPL_REG0` and `RP_RXCPL_REG1` to retrieve dword 2 and dword 3 of the TLP, then repeat step 3.
   - If `RP_RXCPL_STATUS.EOP` = 1'b1, read `RP_RXCPL_REG0` and `RP_RXCPL_REG1` to retrieve final dwords of TLP.

PCI Express to Avalon-MM Interrupt Status and Enable Registers for Root Ports

The Root Port supports MSI, MSI-X and legacy (INTx) interrupts. MSI and MSI-X interrupts are memory writes from the Endpoint to the Root Port. MSI and MSI-X requests are forwarded to the interconnect without asserting `CraIrq_o`.

Table 6-24: Avalon-MM Interrupt Status Registers for Root Ports, 0x3060

Bits	Name	Access Mode	Description
[31:5]	Reserved	—	—
[4]	`RPRX_CPL_RECEIVED`	RW1C	Set to 1'b1 when the Root Port has received a Completion TLP for an outstanding Non-Posted request from the TLP Direct channel.
[3]	`INTD_RECEIVED`	RW1C	The Root Port has received INTD from the Endpoint.
[2]	`INTC_RECEIVED`	RW1C	The Root Port has received INTC from the Endpoint.
[1]	`INTB_RECEIVED`	RW1C	The Root Port has received INTB from the Endpoint.
[0]	`INTA_RECEIVED`	RW1C	The Root Port has received INTA from the Endpoint.

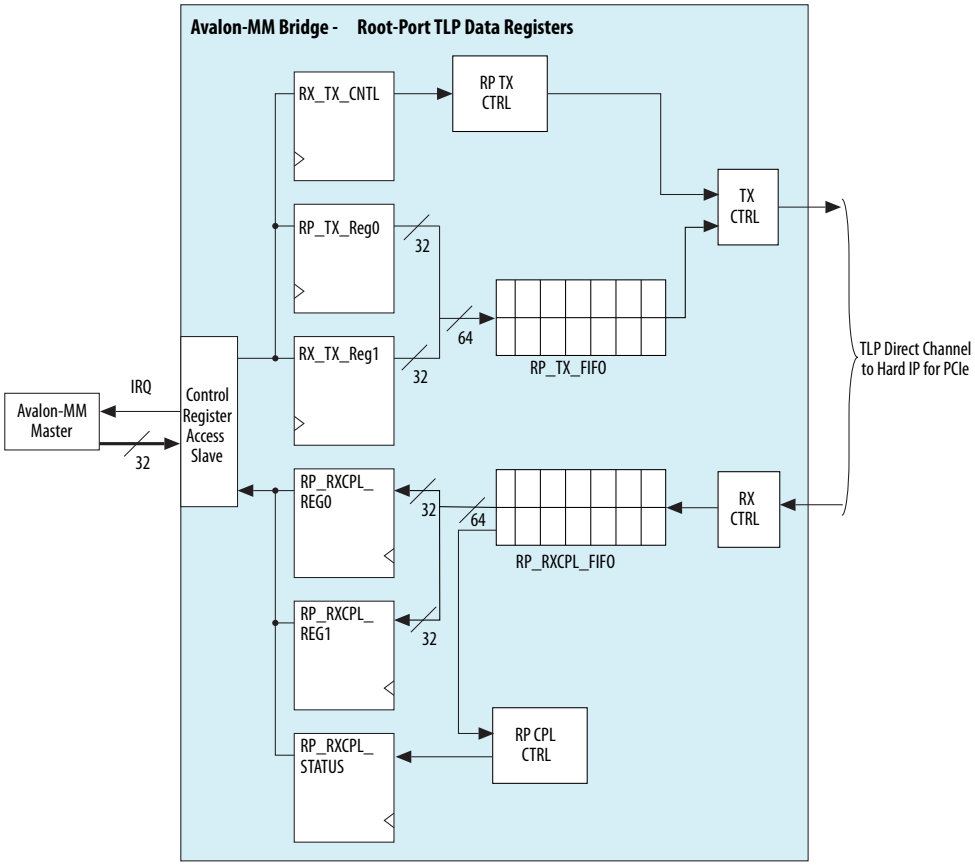
Table 6-25: INT-X Interrupt Enable Register for Root Ports, 0x3070

Bit	Name	Access Mode	Description
[31:5]	Reserved	—	—
[4]	RPRX_CPL_RECEIVED	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register RPRX_CPL_RECEIVED bit indicates it has received a Completion for a Non-Posted request from the TLP Direct channel.
[3]	INTD_RECEIVED_ENA	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register INTD_RECEIVED bit indicates it has received INTD.
[2]	INTC_RECEIVED_ENA	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register INTC_RECEIVED bit indicates it has received INTC.
[1]	INTB_RECEIVED_ENA	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register INTB_RECEIVED bit indicates it has received INTB.
[0]	INTA_RECEIVED_ENA	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register INTA_RECEIVED bit indicates it has received INTA.

Root Port TLP Data Registers

The TLP data registers provide a mechanism for the Application Layer to specify data that the Root Port uses to construct Configuration TLPs, I/O TLPs, and single dword Memory Reads and Write requests. The Root Port then drives the TLPs on the TLP Direct Channel to access the Configuration Space, I/O space, or Endpoint memory.

Figure 6-12: Root Port TLP Data Registers



Note: The high performance TLPs implemented by Avalon-MM ports in the Avalon-MM Bridge are also available for Root Ports. For more information about these TLPs, refer to *Avalon-MM Bridge TLPs*.

Table 6-26: Root Port TLP Data Registers, 0x2000–0x2FFF

Root-Port Request Registers				Address Range: 0x2800-0x2018
Address	Bits	Name	Access	Description
0x2000	[31:0]	RP_TX_REG0	W	Lower 32 bits of the TX TLP.
0x2004	[31:0]	RP_TX_REG1	W	Upper 32 bits of the TX TLP.
0x2008	[31:2]	Reserved	—	—
	[1]	RP_TX_CNTRL.EOP	W	Write 1'b1 to specify the of end a packet. Writing this bit frees the corresponding entry in the FIFO.
	[0]	RP_TX_CNTRL.SOP	W	Write 1'b1 to specify the start of a packet.

Root-Port Request Registers				Address Range: 0x2800-0x2018
Address	Bits	Name	Access	Description
0x2010	[31:16]	Reserved	—	—
	[15:8]	RP_RXCPL_STATUS	R	Specifies the number of words in the RX completion FIFO that contain valid data.
	[7:2]	Reserved	—	—
	[1]	RP_RXCPL_STATUS.EOP	R	When 1'b1, indicates that the data for a Completion TLP is ready to be read by the Application Layer. The Application Layer must poll this bit to determine when a Completion TLP is available.
	[0]	RP_RXCPL_STATUS.SOP	R	When 1'b1, indicates that the final data for a Completion TLP is ready to be read by the Application Layer. The Application Layer must poll this bit to determine when the final data for a Completion TLP is available.
0x2014	[31:0]	RP_RXCPL_REG0	RC	Lower 32 bits of a Completion TLP. Reading frees this entry in the FIFO.
0x2018	[31:0]	RP_RXCPL_REG1	RC	Upper 32 bits of a Completion TLP. Reading frees this entry in the FIFO.

Related Information

Avalon-MM Bridge TLPs on page 10-11

Uncorrectable Internal Error Mask Register

Table 6-27: Uncorrectable Internal Error Mask Register

The `Uncorrectable Internal Error Mask` register controls which errors are forwarded as internal uncorrectable errors. With the exception of the configuration error detected in CvP mode, all of the errors are severe and may place the device or PCIe link in an inconsistent state. The configuration error detected in CvP mode may be correctable depending on the design of the programming software. The access code *RWS* stands for Read Write Sticky meaning the value is retained after a soft reset of the IP core.

Bits	Register Description	Reset Value	Access
[31:12]	Reserved.	1b'0	RO

Bits	Register Description	Reset Value	Access
[11]	Mask for RX buffer posted and completion overflow error.	1b'1	RWS
[10]	Reserved	1b'0	RO
[9]	Mask for parity error detected on Configuration Space to TX bus interface.	1b'1	RWS
[8]	Mask for parity error detected on the TX to Configuration Space bus interface.	1b'1	RWS
[7]	Mask for parity error detected at TX Transaction Layer error.	1b'1	RWS
[6]	Reserved	1b'0	RO
[5]	Mask for configuration errors detected in CvP mode.	1b'0	RWS
[4]	Mask for data parity errors detected during TX Data Link LCRC generation.	1b'1	RWS
[3]	Mask for data parity errors detected on the RX to Configuration Space Bus interface.	1b'1	RWS
[2]	Mask for data parity error detected at the input to the RX Buffer.	1b'1	RWS
[1]	Mask for the retry buffer uncorrectable ECC error.	1b'1	RWS
[0]	Mask for the RX buffer uncorrectable ECC error.	1b'1	RWS

Uncorrectable Internal Error Status Register

Table 6-28: Uncorrectable Internal Error Status Register

This register reports the status of the internally checked errors that are uncorrectable. When specific errors are enabled by the `Uncorrectable Internal Error Mask` register, they are handled as Uncorrectable Internal Errors as defined in the *PCI Express Base Specification 3.0*. This register is for debug only. It should only be used to observe behavior, not to drive custom logic. The access code RW1CS represents Read Write 1 to Clear Sticky.

Bits	Register Description	Reset Value	Access
[31:12]	Reserved.	0	RO
[11]	When set, indicates an RX buffer overflow condition in a posted request or Completion	0	RW1CS

Bits	Register Description	Reset Value	Access
[10]	Reserved.	0	RO
[9]	When set, indicates a parity error was detected on the Configuration Space to TX bus interface	0	RW1CS
[8]	When set, indicates a parity error was detected on the TX to Configuration Space bus interface	0	RW1CS
[7]	When set, indicates a parity error was detected in a TX TLP and the TLP is not sent.	0	RW1CS
[6]	When set, indicates that the Application Layer has detected an uncorrectable internal error.	0	RW1CS
[5]	When set, indicates a configuration error has been detected in CvP mode which is reported as uncorrectable. This bit is set whenever a `CVP_CONFIG_ERROR` rises while in `CVP_MODE`.	0	RW1CS
[4]	When set, indicates a parity error was detected by the TX Data Link Layer.	0	RW1CS
[3]	When set, indicates a parity error has been detected on the RX to Configuration Space bus interface.	0	RW1CS
[2]	When set, indicates a parity error was detected at input to the RX Buffer.	0	RW1CS
[1]	When set, indicates a retry buffer uncorrectable ECC error.	0	RW1CS
[0]	When set, indicates a RX buffer uncorrectable ECC error.	0	RW1CS

Related Information

PCI Express Base Specification 3.0

Correctable Internal Error Mask Register

Table 6-29: Correctable Internal Error Mask Register

The `Correctab le Internal Error Mask` register controls which errors are forwarded as Internal Correctable Errors. This register is for debug only.

Bits	Register Description	Reset Value	Access
[31:7]	Reserved.	0	RO

Bits	Register Description	Reset Value	Access
[6]	Mask for Corrected Internal Error reported by the Application Layer.	1	RWS
[5]	Mask for configuration error detected in CvP mode.	0	RWS
[4:2]	Reserved.	0	RO
[1]	Mask for retry buffer correctable ECC error.	1	RWS
[0]	Mask for RX Buffer correctable ECC error.	1	RWS

Correctable Internal Error Status Register

Table 6-30: Correctable Internal Error Status Register

The `Correctable Internal Error Status` register reports the status of the internally checked errors that are correctable. When these specific errors are enabled by the `Correctable Internal Error Mask` register, they are forwarded as Correctable Internal Errors as defined in the *PCI Express Base Specification 3.0*. This register is for debug only. It should only be used to observe behavior, not to drive logic custom logic.

Bits	Register Description	Reset Value	Access
[31:6]	Reserved.	0	RO
[5]	When set, indicates a configuration error has been detected in CvP mode which is reported as correctable. This bit is set whenever a `CVP_CONFIG_ERROR` occurs while in `CVP_MODE`.	0	RW1CS
[4:2]	Reserved.	0	RO
[1]	When set, the retry buffer correctable ECC error status indicates an error.	0	RW1CS
[0]	When set, the RX buffer correctable ECC error status indicates an error.	0	RW1CS

Related Information

PCI Express Base Specification 3.0

Arria 10 Reset and Clocks

7

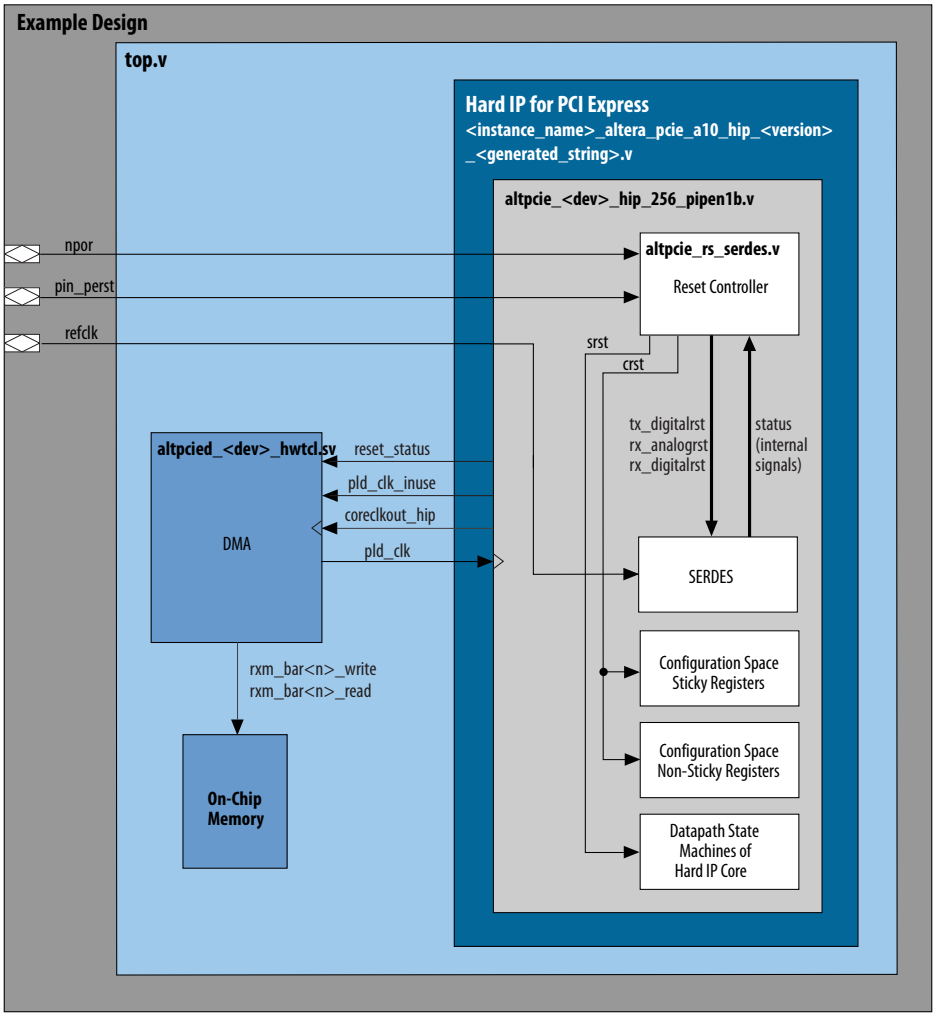
2015.06.05

UG-01145_avmm

Subscribe

Send Feedback

Figure 7-1: Reset Controller in Arria 10 Devices



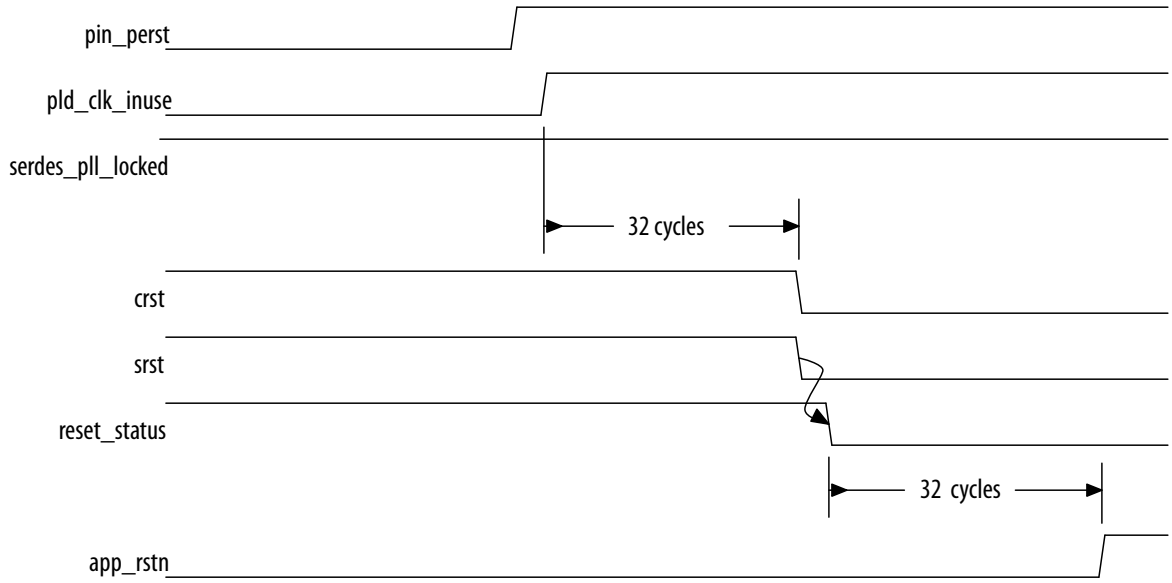
© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Reset Sequence for Hard IP for PCI Express IP Core and Application Layer

Figure 7-2: Hard IP for PCI Express and Application Logic Reset Sequence

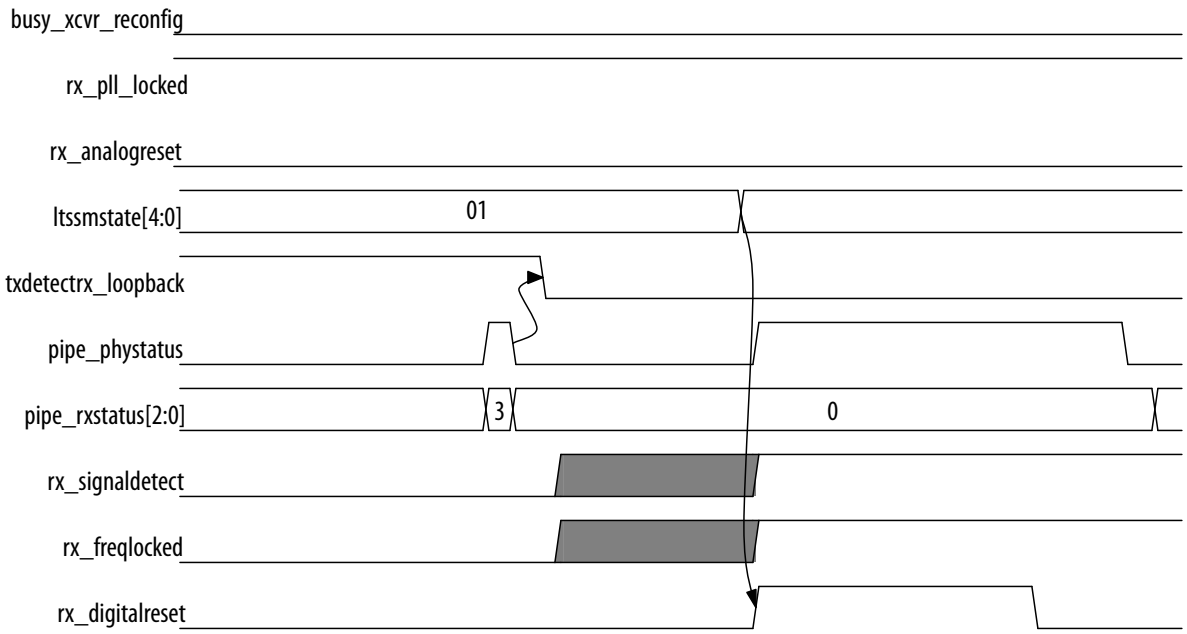
Your Application Layer can instantiate a module similar to the one in this figure to generate `app_rstn`, which resets the Application Layer logic.



This reset sequence includes the following steps:

1. After `pin_perst` or `npor` is released, the Hard IP reset controller waits for `pld_clk_inuse` to be asserted.
2. `csrt` and `srst` are released 32 cycles after `pld_clk_inuse` is asserted.
3. The Hard IP for PCI Express deasserts the `reset_status` output to the Application Layer.
4. The **altpcied_<*device*>v_hwtcl.sv** deasserts `app_rstn` 32 `pld_clk`cycles after `reset_status` is released.

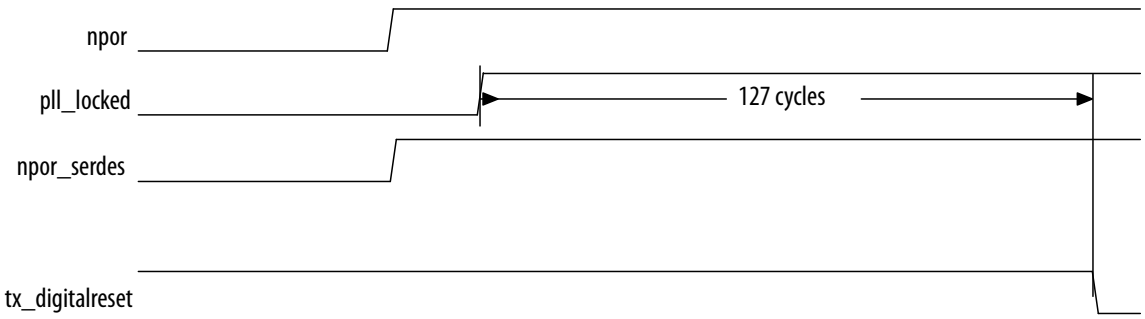
Figure 7-3: RX Transceiver Reset Sequence



The RX transceiver reset sequence includes the following steps:

1. After `rx_pll_locked` is asserted, the LTSSM state machine transitions from the Detect.Quiet to the Detect.Active state.
2. When the `pipe_phystatus` pulse is asserted and `pipe_rxstatus[2:0]` = 3, the receiver detect operation has completed.
3. The LTSSM state machine transitions from the Detect.Active state to the Polling.Active state.
4. The Hard IP for PCI Express asserts `rx_digitalreset`. The `rx_digitalreset` signal is deasserted after `rx_signaldetect` is stable for a minimum of 3 ms.

Figure 7-4: TX Transceiver Reset Sequence



The TX transceiver reset sequence includes the following steps:

1. After `npor` is deasserted, the IP core deasserts the `npor_serdes` input to the TX transceiver.
2. The SERDES reset controller waits for `pll_locked` to be stable for a minimum of 127 `pld_clk` cycles before deasserting `tx_digitalreset`.

For descriptions of the available reset signals, refer to *Reset Signals, Status, and Link Training Signals*.

Clocks

The Hard IP contains a clock domain crossing (CDC) synchronizer at the interface between the PHY/MAC and the DLL layers. The synchronizer allows the Data Link and Transaction Layers to run at frequencies independent of the PHY/MAC. The CDC synchronizer provides more flexibility for the user clock interface. Depending on parameters you specify, the core selects the appropriate `coreclkout_hip`. You can use these parameters to enhance performance by running at a higher frequency for latency optimization or at a lower frequency to save power.

In accordance with the *PCI Express Base Specification*, you must provide a 100 MHz reference clock that is connected directly to the transceiver.

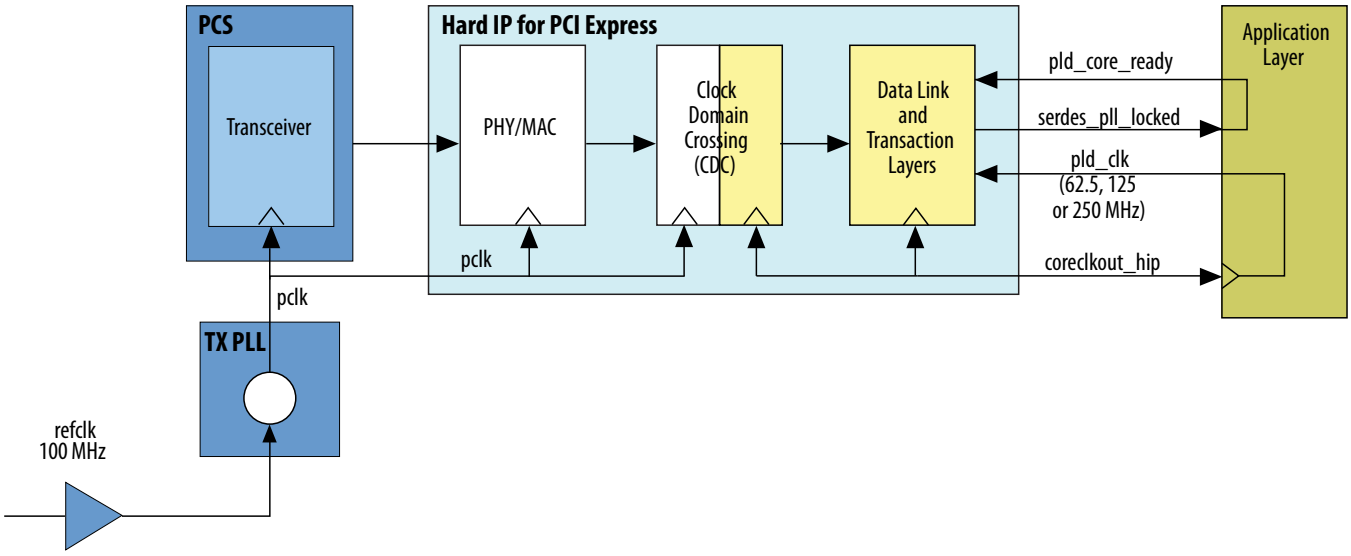
Related Information

PCI Express Base Specification 3.0

Clock Domains

Figure 7-5: Clock Domains and Clock Generation for the Application Layer

The following illustrates the clock domains when using `coreclkout_hip` to drive the Application Layer and the `pld_clk` of the IP core. The Altera-provided example design connects `coreclkout_hip` to the `pld_clk`. However, this connection is not mandatory.



As this figure indicates, the IP core includes the following clock domains:

coreclkout_hip

Table 7-1: Application Layer Clock Frequency for All Combinations of Link Width, Data Rate and Application Layer Interface Widths

The `coreclkout_hip` signal is derived from `pclk`. The following table lists frequencies for `coreclkout_hip`, which are a function of the link width, data rate, and the width of the Application Layer to Transaction Layer interface. The frequencies and widths specified in this table are maintained throughout operation. If the link downtrains to a lesser link width or changes to a different maximum link rate, it maintains the frequencies it was originally configured for as specified in this table. (The Hard IP throttles the interface to achieve a lower throughput.)

Link Width	Maximum Link Rate	Avalon Interface Width	coreclkout_hip
×1	Gen1	64	62.5 MHz(3)
×1	Gen1	64	125 MHz
×2	Gen1	64	125 MHz
×4	Gen1	64	125 MHz
×8	Gen1	64	250 MHz
×8	Gen1	128	125 MHz
×1	Gen2	64	125 MHz
×2	Gen2	64	125 MHz
×4	Gen2	64	250 MHz
×4	Gen2	128	125 MHz
×8	Gen2	128	250 MHz
×8	Gen2	256	125 MHz
×1	Gen3	64	125 MHz
×2	Gen3	64	125 MHz
×2	Gen3	128	250 MHz
×4	Gen3	128	250 MHz

(3) This mode saves power

Link Width	Maximum Link Rate	Avalon Interface Width	coreclkout_hip
×4	Gen3	256	125 MHz
×8	Gen3	256	250 MHz

pld_clk

`coreclkout_hip` can drive the Application Layer clock along with the `pld_clk` input to the IP core. The `pld_clk` can optionally be sourced by a different clock than `coreclkout_hip`. The `pld_clk` minimum frequency cannot be lower than the `coreclkout_hip` frequency. Based on specific Application Layer constraints, a PLL can be used to derive the desired frequency.

Clock Summary

Table 7-2: Clock Summary

Name	Frequency	Clock Domain
`coreclkout_hip`	62.5, 125 or 250 MHz	Avalon-ST interface between the Transaction and Application Layers.
`pld_clk`	62.5, 125, or 250 MHz	Application and Transaction Layers.
`refclk`	100 MHz	SERDES (transceiver). Dedicated free running input clock to the SERDES block.

Interrupts for Endpoints

8

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

The PCI Express Avalon-MM bridge supports MSI or legacy interrupts. The completer only single dword variant includes an interrupt handler that implements both INTX and MSI interrupts. Support requires instantiation of the CRA slave module where the interrupt registers and control logic are implemented.

The PCI Express Avalon-MM bridge supports the Avalon-MM individual requests interrupt scheme: multiple input signals indicate incoming interrupt requests, and software must determine priorities for servicing simultaneous interrupts.

The RX master module port has up to 16 Avalon-MM interrupt input signals (`RXmirq_irq[ <n> :0]`, where <*n*> ≤15). Each interrupt signal indicates a distinct interrupt source. Assertion of any of these signals, or a PCI Express mailbox register write access, sets a bit in the `Avalon-MM to PCI Express Interrupt Status` register. Multiple bits can be set at the same time; Application Layer software on the host side determines priorities for servicing simultaneous incoming interrupt requests. Each set bit in the `Avalon-MM to PCI Express Interrupt Status` register generates a PCI Express interrupt, if enabled, when software determines its turn. Software can enable the individual interrupts by writing to the `Avalon-MM to PCI Express Interrupt Enable Register` through the CRA slave.

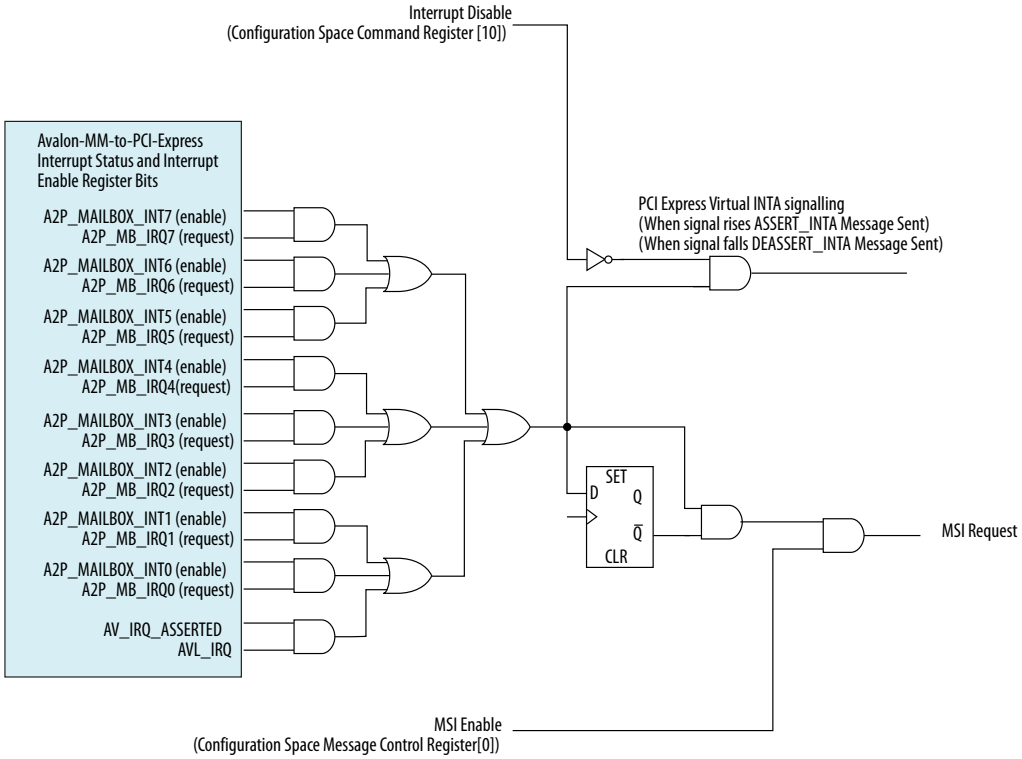
When any interrupt input signal is asserted, the corresponding bit is written in the `Avalon-MM to PCI Express Interrupt Status Register`. Software reads this register and decides priority on servicing requested interrupts.

After servicing the interrupt, software must clear the appropriate serviced interrupt `status` bit and ensure that no other interrupts are pending. For interrupts caused by `Avalon-MM to PCI Express Interrupt Status Register` mailbox writes, the status bits should be cleared in the `Avalon-MM to PCI Express Interrupt Status Register`. For interrupts due to the incoming interrupt signals on the Avalon-MM interface, the interrupt status should be cleared in the Avalon-MM component that sourced the interrupt. This sequence prevents interrupt requests from being lost during interrupt servicing.

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure 8-1: Avalon-MM Interrupt Propagation to the PCI Express Link



Related Information

- **Avalon-MM to PCI Express Interrupt Enable Registers** on page 6-16
- **Avalon-MM to PCI Express Interrupt Status Registers** on page 6-15

Enabling MSI or Legacy Interrupts

The PCI Express Avalon-MM bridge selects either MSI or legacy interrupts automatically based on the standard interrupt controls in the PCI Express Configuration Space registers. Software can write the `Interrupt Disable` bit, which is bit 10 of the `Command` register (at Configuration Space offset 0x4) to disable legacy interrupts. Software can write the `MSI Enable` bit, which is bit 0 of the `MSI Control Status` register in the MSI capability register (bit 16 at configuration space offset 0x50), to enable MSI interrupts.

Software can only enable one type of interrupt at a time. However, to change the selection of MSI or legacy interrupts during operation, software must ensure that no interrupt request is dropped. Therefore, software must first enable the new selection and then disable the old selection. To set up legacy interrupts, software must first clear the `Interrupt Disable` bit and then clear the `MSI enable` bit. To set up MSI interrupts, software must first set the `MSI enable` bit and then set the `Interrupt Disable` bit.

Generation of Avalon-MM Interrupts

The generation of Avalon-MM interrupts requires the instantiation of the CRA slave module where the interrupt registers and control logic are implemented. The CRA slave port has an Avalon-MM Interrupt output signal, `cra_Irq_irq`. A write access to an Avalon-MM mailbox register sets one of the `P2A_MAILBOX_INT<n>` bits in the `Avalon-MM to PCI Express Interrupt Status Register` and asserts the `cra_Irq_o` or `cra_Irq_irq` output, if enabled. Software can enable the interrupt by writing to the `INT_X Interrupt Enable Register for Endpoints` through the CRA slave. After servicing the interrupt, software must clear the appropriate serviced interrupt `status` bit in the PCI-Express-to-Avalon-MM `Interrupt Status` register and ensure that no other interrupt is pending.

Related Information

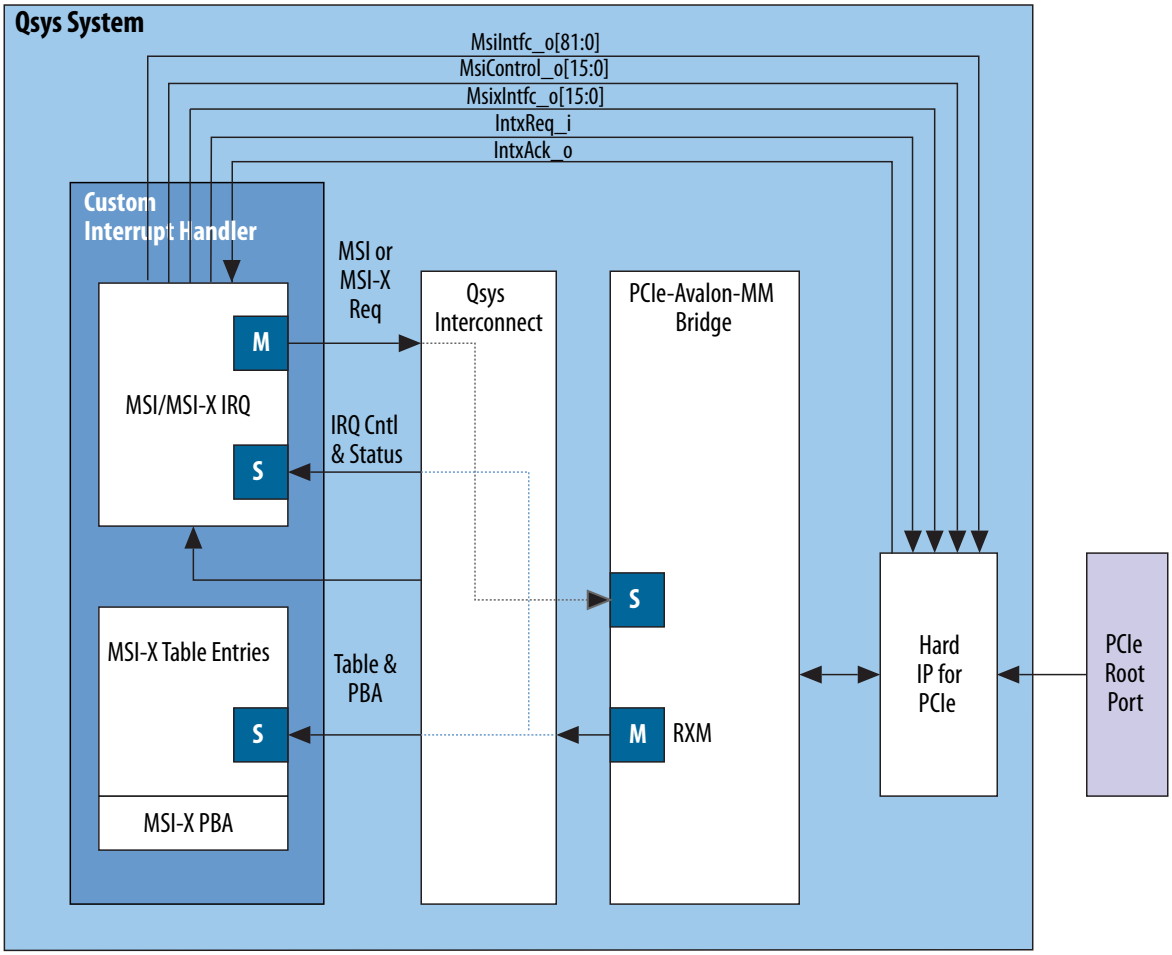
- **Avalon-MM to PCI Express Interrupt Status Registers** on page 6-15
- **PCI Express to Avalon-MM Interrupt Status and Enable Registers for Endpoints** on page 6-19

Interrupts for Endpoints Using the Avalon-MM Interface with Multiple MSI/MSI-X Support

If you select **Enable multiple MSI/MSI X support** under the **Avalon-MM System Settings** banner in the parameter editor, the Hard IP for PCI Express exports the MSI, MSI-X, and INTx interfaces to the Application Layer. The Application Layer must include a Custom Interrupt Handler to send interrupts to the Root Port. You must design this Custom Interrupt Handler. The following figure provides an overview of the logic for the Custom Interrupt Handler. The Custom Interrupt Handler should include hardware to perform the following tasks:

- An MSI/MXI-X IRQ Avalon-MM Master port to drive MSI or MSI-X interrupts as memory writes to the PCIe Avalon-MM bridge.
- A legacy interrupt signal, `IntxReq_i`, to drive legacy interrupts from the MSI/MSI-X IRQ module to the Hard IP for PCI Express.
- An MSI/MSI-X Avalon-MM Slave port to receive interrupt control and status from the PCIe Root Port.
- An MSI-X table to store the MSI-X table entries. The PCIe Root Port sets up this table.

Figure 8-2: Block Diagram for Custom Interrupt Handler



Refer to *Interrupts for Endpoints* for the definitions of MSI, MSI-X, and INTx buses.

For more information about implementing MSI or MSI-X interrupts, refer to the *PCI Local Bus Specification, Revision 2.3, MSI-X ECN*.

For more information about implementing interrupts, including an MSI design example, refer to *Handling PCIe Interrupts* on the Altera wiki.

Related Information

- **Interrupts for Endpoints** on page 8-1
- **PCI Local Bus Specification, Revision 2.3**
- **Handling PCIe Interrupts**

Error Handling 9

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

Each PCI Express compliant device must implement a basic level of error management and can optionally implement advanced error management. The IP core implements both basic and advanced error reporting. Error handling for a Root Port is more complex than that of an Endpoint.

Table 9-1: Error Classification

The *PCI Express Base Specification* defines three types of errors, outlined in the following table.

Type	Responsible Agent	Description
Correctable	Hardware	While correctable errors may affect system performance, data integrity is maintained.
Uncorrectable, non-fatal	Device software	Uncorrectable, non-fatal errors are defined as errors in which data is lost, but system integrity is maintained. For example, the fabric may lose a particular TLP, but it still works without problems.
Uncorrectable, fatal	System software	Errors generated by a loss of data and system failure are considered uncorrectable and fatal. Software must determine how to handle such errors: whether to reset the link or implement other means to minimize the problem.

Related Information

PCI Express Base Specification 3.0

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Physical Layer Errors

Table 9-2: Errors Detected by the Physical Layer

The following table describes errors detected by the Physical Layer. Physical Layer error reporting is optional in the *PCI Express Base Specification*.

Error	Type	Description

Receive port error	Correctable	This error has the following 3 potential causes:
		• Physical coding sublayer error when a lane is in L0 state. These errors are reported to the Hard IP block via the per lane PIPE interface input receive status signals, `rxstatus<lane_number>[2:0]` using the following encodings:
		  • 3'b100: 8B/10B Decode Error
		  • 3'b101: Elastic Buffer Overflow
		  • 3'b110: Elastic Buffer Underflow
		  • 3'b111: Disparity Error
		• Deskew error caused by overflow of the multilane deskew FIFO.
		• Control symbol received in wrong lane.

Data Link Layer Errors

Table 9-3: Errors Detected by the Data Link Layer

Error	Type	Description
Bad TLP	Correctable	This error occurs when a LCRC verification fails or when a sequence number error occurs.
Bad DLLP	Correctable	This error occurs when a CRC verification fails.
Replay timer	Correctable	This error occurs when the replay timer times out.
Replay num rollover	Correctable	This error occurs when the replay number rolls over.
Data Link Layer protocol	Uncorrectable(fatal)	This error occurs when a sequence number specified by the Ack/Nak block in the Data Link Layer (`AckNak_Seq_Num`) does not correspond to an unacknowledged TLP.

Transaction Layer Errors

Table 9-4: Errors Detected by the Transaction Layer

Error	Type	Description
Poisoned TLP received	Uncorrectable (non-fatal)	This error occurs if a received Transaction Layer packet has the EP poison bit set. The received TLP is passed to the Application Layer and the Application Layer logic must take appropriate action in response to the poisoned TLP. Refer to "2.7.2.2 Rules for Use of Data Poisoning" in the *PCI Express Base Specification* for more information about poisoned TLPs.
ECRC check failed (1)	Uncorrectable (non-fatal)	This error is caused by an ECRC check failing despite the fact that the TLP is not malformed and the LCRC check is valid. The Hard IP block handles this TLP automatically. If the TLP is a non-posted request, the Hard IP block generates a completion with completer abort status. In all cases the TLP is deleted in the Hard IP block and not presented to the Application Layer.
Unsupported Request for Endpoints	Uncorrectable (non-fatal)	This error occurs whenever a component receives any of the following Unsupported Requests: • Type 0 Configuration Requests for a non-existing function. • Completion transaction for which the Requester ID does not match the bus, device and function number. • Unsupported message. • A Type 1 Configuration Request TLP for the TLP from the PCIe link. • A locked memory read (MEMRDLK) on native Endpoint. • A locked completion transaction. • A 64-bit memory transaction in which the 32 MSBs of an address are set to 0. • A memory or I/O transaction for which there is no BAR match. • A memory transaction when the Memory Space Enable bit (bit [1] of the PCI Command register at Configuration Space offset 0x4) is set to 0. • A poisoned configuration write request (`CfgWr0`)

Error	Type	Description
		In all cases the TLP is deleted in the Hard IP block and not presented to the Application Layer. If the TLP is a non-posted request, the Hard IP block generates a completion with Unsupported Request status.
Unsupported Requests for Root Port	Uncorrectable (fatal)	This error occurs whenever a component receives an Unsupported Request including: • Unsupported message • A Type 0 Configuration Request TLP • A 64-bit memory transaction which the 32 MSBs of an address are set to 0. • A memory transaction that does not match the address range defined by the Base and Limit Address registers
Completion timeout	Uncorrectable (non-fatal)	This error occurs when a request originating from the Application Layer does not generate a corresponding completion TLP within the established time. It is the responsibility of the Application Layer logic to provide the completion timeout mechanism. The completion timeout should be reported from the Transaction Layer using the `cpl_err[0]` signal.
Completer abort *(1)*	Uncorrectable (non-fatal)	The Application Layer reports this error using the `cpl_err[2]`signal when it aborts receipt of a TLP.

Error	Type	Description
Unexpected completion	Uncorrectable (non-fatal)	This error is caused by an unexpected completion transaction. The Hard IP block handles the following conditions: • The Requester ID in the completion packet does not match the Configured ID of the Endpoint. • The completion packet has an invalid tag number. (Typically, the tag used in the completion packet exceeds the number of tags specified.) • The completion packet has a tag that does not match an outstanding request. • The completion packet for a request that was to I/O or Configuration Space has a length greater than 1 dword. • The completion status is Configuration Retry Status (CRS) in response to a request that was not to Configuration Space. In all of the above cases, the TLP is not presented to the Application Layer; the Hard IP block deletes it. The Application Layer can detect and report other unexpected completion conditions using the `cpl_err[2]` signal. For example, the Application Layer can report cases where the total length of the received successful completions do not match the original read request length.
Receiver overflow *(1)*	Uncorrectable (fatal)	This error occurs when a component receives a TLP that violates the FC credits allocated for this type of TLP. In all cases the hard IP block deletes the TLP and it is not presented to the Application Layer.
Flow control protocol error (FCPE) *(1)*	Uncorrectable (fatal)	This error occurs when a component does not receive update flow control credits with the 200 µs limit.

Error	Type	Description
Malformed TLP	Uncorrectable (fatal)	This error is caused by any of the following conditions: • The data payload of a received TLP exceeds the maximum payload size. • The TD field is asserted but no TLP digest exists, or a TLP digest exists but the TD bit of the PCI Express request header packet is not asserted. • A TLP violates a byte enable rule. The Hard IP block checks for this violation, which is considered optional by the PCI Express specifications. • A TLP in which the type and length fields do not correspond with the total length of the TLP. • A TLP in which the combination of format and type is not specified by the PCI Express specification. • A request specifies an address/length combination that causes a memory space access to exceed a 4 KByte boundary. The Hard IP block checks for this violation, which is considered optional by the PCI Express specification. • Messages, such as Assert_INTX, Power Management, Error Signaling, Unlock, and Set Power Slot Limit, must be transmitted across the default traffic class. The Hard IP block deletes the malformed TLP; it is not presented to the Application Layer.

Note:

1. Considered optional by the *PCI Express Base Specification Revision* .

Error Reporting and Data Poisoning

How the Endpoint handles a particular error depends on the configuration registers of the device.

Refer to the *PCI Express Base Specification 3.0* for a description of the device signaling and logging for an Endpoint.

The Hard IP block implements data poisoning, a mechanism for indicating that the data associated with a transaction is corrupted. Poisoned TLPs have the error/poisoned bit of the header set to 1 and observe the following rules:

- Received poisoned TLPs are sent to the Application Layer and status bits are automatically updated in the Configuration Space.
- Received poisoned Configuration Write TLPs are not written in the Configuration Space.
- The Configuration Space never generates a poisoned TLP; the error/poisoned bit of the header is always set to 0.

Poisoned TLPs can also set the parity error bits in the PCI Configuration Space Status register.

Table 9-5: Parity Error Conditions

Status Bit	Conditions
Detected parity error (status register bit 15)	Set when any received TLP is poisoned.
Master data parity error (status register bit 8)	This bit is set when the command register parity enable bit is set and one of the following conditions is true: • The poisoned bit is set during the transmission of a Write Request TLP. • The poisoned bit is set on a received completion TLP.

Poisoned packets received by the Hard IP block are passed to the Application Layer. Poisoned transmit TLPs are similarly sent to the link.

Related Information

PCI Express Base Specification 3.0

Uncorrectable and Correctable Error Status Bits

The following section is reprinted with the permission of PCI-SIG. Copyright 2010 PCI-SIG.

Figure 9-1: Uncorrectable Error Status Register

The default value of all the bits of this register is 0. An error status bit that is set indicates that the error condition it represents has been detected. Software may clear the error status by writing a 1 to the appropriate bit.

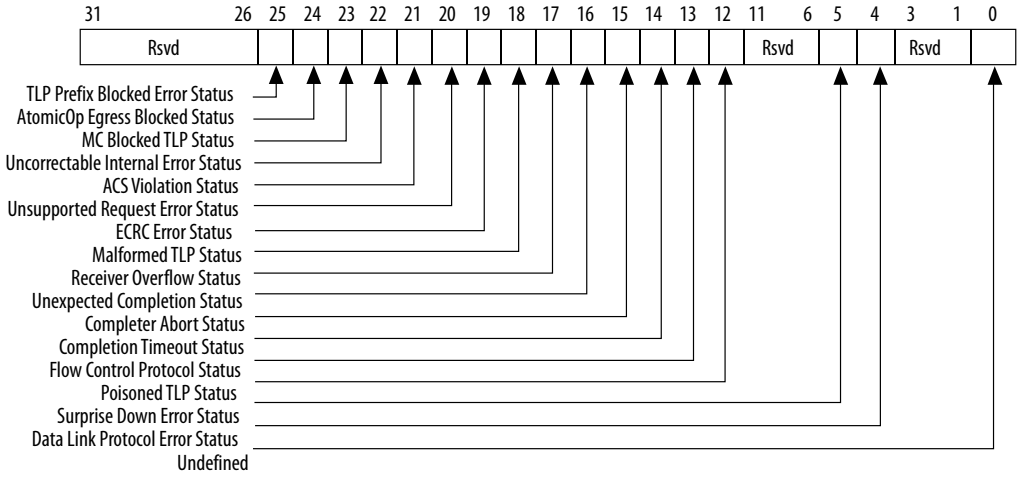
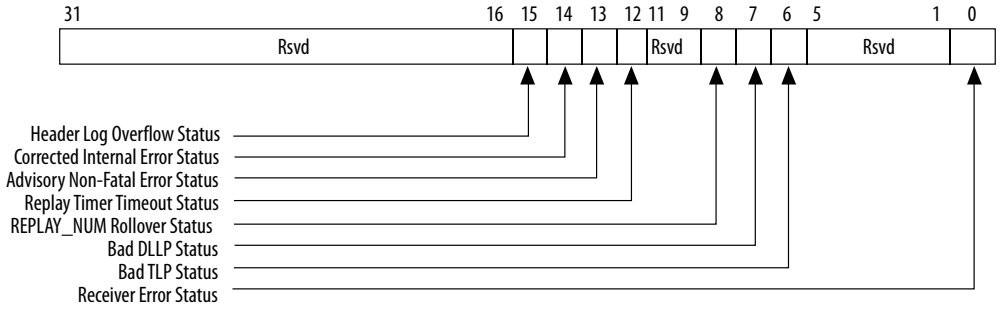


Figure 9-2: Correctable Error Status Register

The default value of all the bits of this register is 0. An error status bit that is set indicates that the error condition it represents has been detected. Software may clear the error status by writing a 1 to the appropriate bit.



IP Core Architecture 10

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

The Avalon-MM Arria 10 Hard IP for PCI Express implements the complete PCI Express protocol stack as defined in the *PCI Express Base Specification.* The protocol stack includes the following layers:

- *Transaction Layer*—The Transaction Layer contains the Configuration Space, which manages communication with the Application Layer, the RX and TX channels, the RX buffer, and flow control credits.
- *Data Link Layer*—The Data Link Layer, located between the Physical Layer and the Transaction Layer, manages packet transmission and maintains data integrity at the link level. Specifically, the Data Link Layer performs the following tasks:
  - Manages transmission and reception of Data Link Layer Packets (DLLPs)
  - Generates all transmission cyclical redundancy code (CRC) values and checks all CRCs during reception
  - Manages the retry buffer and retry mechanism according to received ACK/NAK Data Link Layer packets
  - Initializes the flow control mechanism for DLLPs and routes flow control credits to and from the Transaction Layer
- *Physical Layer*—The Physical Layer initializes the speed, lane numbering, and lane width of the PCI Express link according to packets received from the link and directives received from higher layers.

The following figure provides a high-level block diagram.

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure 10-1: Arria 10 Hard IP for PCI Express Using the Avalon-MM Interface

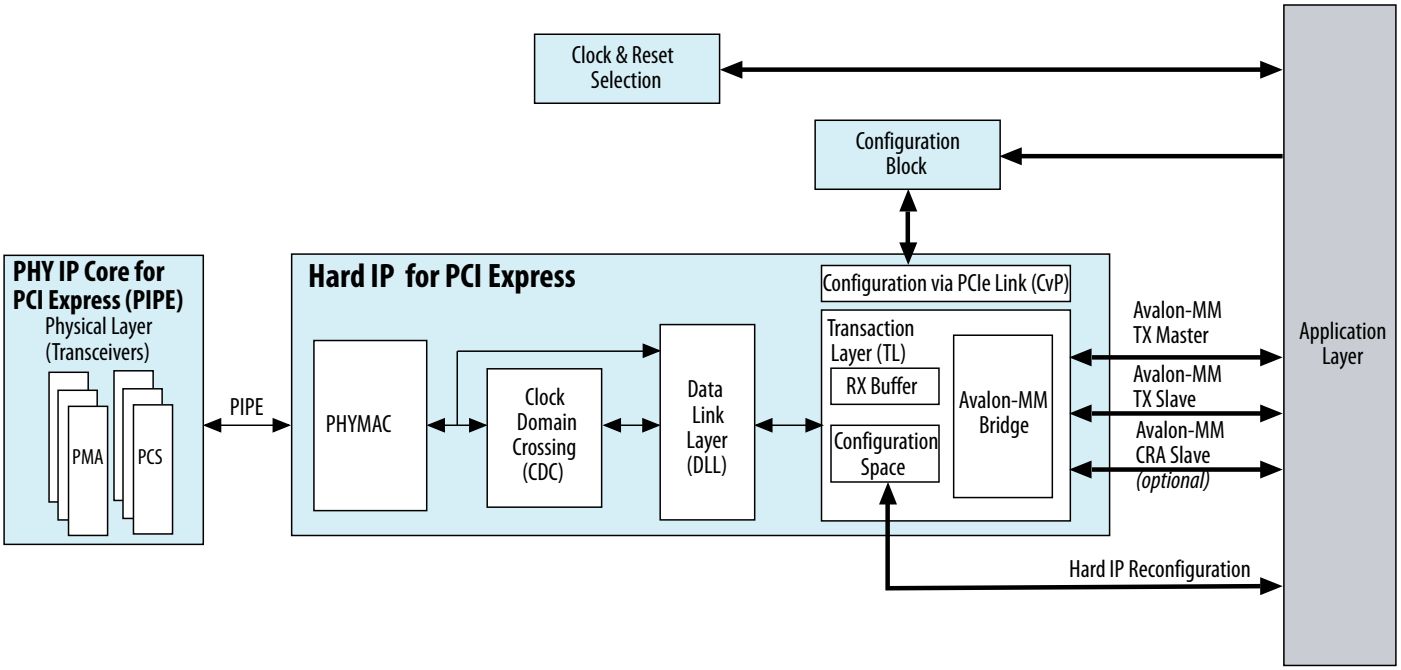


Table 10-1: Application Layer Clock Frequencies

Lanes	Gen1	Gen2	Gen3
×1	125 MHz @ 64 bits or 62.5 MHz @ 64 bits	125 MHz @ 64 bits	125 MHz @64 bits
×2	125 MHz @ 64 bits	125 MHz @ 128 bits	250 MHz @ 64 bits or 125 MHz @ 128 bits
×4	125 MHz @ 64 bits	250 MHz @ 64 bits or 125 MHz @ 128 bits	250 MHz @ 128 bits or 125 MHz @ 256 bits
×8	250 MHz @ 64 bits or 125 MHz @ 128 bits	250 MHz @ 128 bits or 125 MHz @ 256 bits	250 MHz @ 256 bits

Related Information

PCI Express Base Specification 3.0

Top-Level Interfaces

Avalon-MM Interface

An Avalon-MM interface connects the Application Layer and the Transaction Layer. The Avalon-MM interface implement the Avalon-MM protocol described in the *Avalon Interface Specifications*. Refer to this specification for information about the Avalon-MM protocol, including timing diagrams.

Related Information

- **64- or 128-Bit Avalon-MM Interface to the Application Layer** on page 5-1
- **Avalon Interface Specifications**

Clocks and Reset

The *PCI Express Base Specification* requires an input reference clock, which is called `refclk` in this design. The *PCI Express Base Specification* stipulates that the frequency of this clock be 100 MHz.

The *PCI Express Base Specification* also requires a system configuration time of 100 ms. To meet this specification, IP core includes an embedded hard reset controller. This reset controller exits the reset state after the I/O ring of the device is initialized.

Interrupts

The Hard IP for PCI Express offers the following interrupt mechanisms:

- Message Signaled Interrupts (MSI)— MSI uses the Transaction Layer's request-acknowledge handshaking protocol to implement interrupts. The MSI Capability structure is stored in the Configuration Space and is programmable using Configuration Space accesses.
- MSI-X—The Transaction Layer generates MSI-X messages which are single dword memory writes. In contrast to the MSI capability structure, which contains all of the control and status information for the interrupt vectors, the MSI-X Capability structure points to an MSI-X table structure and MSI-X PBA structure which are stored in memory.

Related Information

Interrupts for Endpoints when Multiple MSI/MSI-X Support Is Enabled on page 5-10

PIPE

The PIPE interface implements the Intel-designed PIPE interface specification. You can use this parallel interface to speed simulation; however, you cannot use the PIPE interface in actual hardware.

- The Gen1, Gen2, and Gen3 simulation models support PIPE and serial simulation.
- For Gen3, the Altera BFM bypasses Gen3 Phase 2 and Phase 3 Equalization. However, Gen3 variants can perform Phase 2 and Phase 3 equalization if instructed by a third-party BFM.

Related Information

PIPE Interface Signals on page 5-14

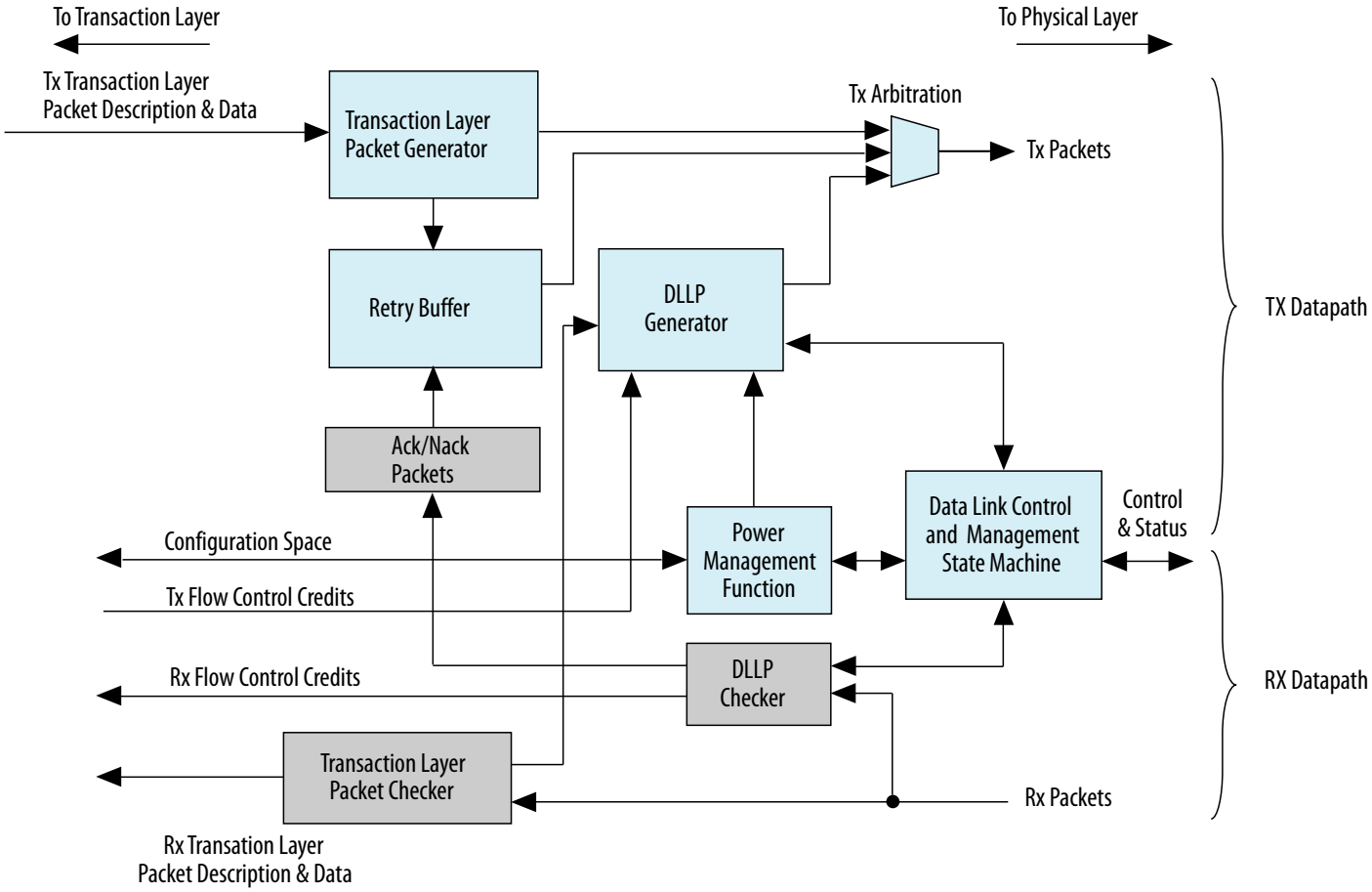
Data Link Layer

The Data Link Layer is located between the Transaction Layer and the Physical Layer. It maintains packet integrity and communicates (by DLL packet transmission) at the PCI Express link level (as opposed to component communication by TLP transmission in the interconnect fabric).

The DLL implements the following functions:

- Link management through the reception and transmission of DLL packets (DLLP), which are used for the following functions:
  - Power management of DLLP reception and transmission
  - To transmit and receive ACK/NACK packets
  - Data integrity through generation and checking of CRCs for TLPs and DLLPs
  - TLP retransmission in case of NAK DLLP reception using the retry buffer
  - Management of the retry buffer
  - Link retraining requests in case of error through the Link Training and Status State Machine (LTSSM) of the Physical Layer

Figure 10-2: Data Link Layer



The DLL has the following sub-blocks:

- Data Link Control and Management State Machine—This state machine is synchronized with the Physical Layer's LTSSM state machine and is also connected to the Configuration Space Registers. It initializes the link and flow control credits and reports status to the Configuration Space.
- Power Management—This function handles the handshake to enter low power mode. Such a transition is based on register values in the Configuration Space and received Power Management (PM) DLLPs.
- Data Link Layer Packet Generator and Checker—This block is associated with the DLLP's 16-bit CRC and maintains the integrity of transmitted packets.
- Transaction Layer Packet Generator—This block generates transmit packets, generating a sequence number and a 32-bit CRC (LCRC). The packets are also sent to the retry buffer for internal storage. In retry mode, the TLP generator receives the packets from the retry buffer and generates the CRC for the transmit packet.
- Retry Buffer—The retry buffer stores TLPs and retransmits all unacknowledged packets in the case of NAK DLLP reception. In case of ACK DLLP reception, the retry buffer discards all acknowledged packets.

- ACK/NAK Packets—The ACK/NAK block handles ACK/NAK DLLPs and generates the sequence number of transmitted packets.
- Transaction Layer Packet Checker—This block checks the integrity of the received TLP and generates a request for transmission of an ACK/NAK DLLP.
- TX Arbitration—This block arbitrates transactions, prioritizing in the following order:
  - Initialize FC Data Link Layer packet
  - ACK/NAK DLLP (high priority)
  - Update FC DLLP (high priority)
  - PM DLLP
  - Retry buffer TLP
  - TLP
  - Update FC DLLP (low priority)
  - ACK/NAK FC DLLP (low priority)

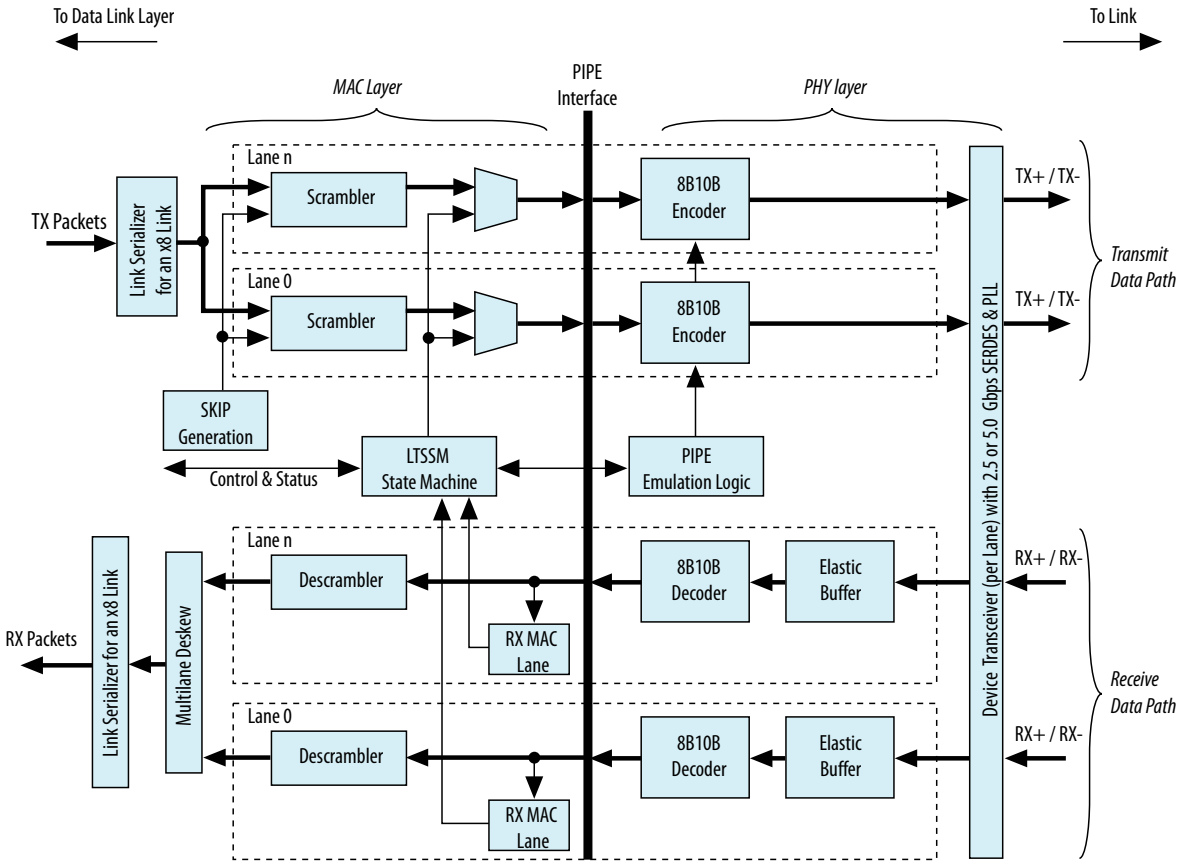
Physical Layer

The Physical Layer is the lowest level of the PCI Express protocol stack. It is the layer closest to the serial link. It encodes and transmits packets across a link and accepts and decodes received packets. The Physical Layer connects to the link through a high-speed SERDES interface running at 2.5 Gbps for Gen1 implementations, at 2.5 or 5.0 Gbps for Gen2 implementations, and at 2.5, 5.0 or 8.0 Gbps for Gen3 implementations.

The Physical Layer is responsible for the following actions:

- Initializing the link
- Scrambling/descrambling and 8B/10B encoding/decoding for 2.5 Gbps (Gen1), 5.0 Gbps (Gen2), or 128b/130b encoding/decoding of 8.0 Gbps (Gen3) per lane
- Serializing and deserializing data
- Operating the PIPE 3.0 Interface
- Implementing auto speed negotiation (Gen2 and Gen3)
- Transmitting and decoding the training sequence
- Providing hardware autonomous speed control
- Implementing auto lane reversal

Figure 10-3: Physical Layer Architecture



The Physical Layer is subdivided by the PIPE Interface Specification into two layers (bracketed horizontally in above figure):

- Media Access Controller (MAC) Layer—The MAC layer includes the LTSSM and the scrambling/descrambling and multilane deskew functions.
- PHY Layer—The PHY layer includes the 8B/10B and 128b/130b encode/decode functions, elastic buffering, and serialization/deserialization functions.

The Physical Layer integrates both digital and analog elements. Intel designed the PIPE interface to separate the MAC from the PHY. The Arria 10 Hard IP for PCI Express complies with the PIPE interface specification.

The PHYMAC block comprises four main sub-blocks:

- MAC Lane—Both the RX and the TX path use this block.
  - On the RX side, the block decodes the Physical Layer packet and reports to the LTSSM the type and number of TS1/TS2 ordered sets received.
  - On the TX side, the block multiplexes data from the DLL and the LTSTX sub-block. It also adds lane specific information, including the lane number and the force PAD value when the LTSSM disables the lane during initialization.
- LTSSM—This block implements the LTSSM and logic that tracks TX and RX data on each lane.
- For transmission, it interacts with each MAC lane sub-block and with the LTSTX sub-block by asserting both global and per-lane control bits to generate specific Physical Layer packets.
  - On the receive path, it receives the Physical Layer packets reported by each MAC lane sub-block. It also enables the multilane deskew block. This block reports the Physical Layer status to higher layers.
  - LTSTX (Ordered Set and SKP Generation)—This sub-block generates the Physical Layer packet. It receives control signals from the LTSSM block and generates Physical Layer packet for each lane. It generates the same Physical Layer Packet for all lanes and PAD symbols for the link or lane number in the corresponding TS1/TS2 fields.

The block also handles the receiver detection operation to the PCS sub-layer by asserting predefined PIPE signals and waiting for the result. It also generates a SKP Ordered Set at every predefined timeslot and interacts with the TX alignment block to prevent the insertion of a SKP Ordered Set in the middle of packet.

- Deskew—This sub-block performs the multilane deskew function and the RX alignment between the number of initialized lanes and the 64-bit data path.

The multilane deskew implements an eight-word FIFO buffer for each lane to store symbols. Each symbol includes eight data bits, one disparity bit, and one control bit. The FIFO discards the FTS, COM, and SKP symbols and replaces PAD and IDL with D0.0 data. When all eight FIFOs contain data, a read can occur.

When the multilane lane deskew block is first enabled, each FIFO begins writing after the first COM is detected. If all lanes have not detected a COM symbol after seven clock cycles, they are reset and the resynchronization process restarts, or else the RX alignment function recreates a 64-bit data word which is sent to the DLL.

32-Bit PCI Express Avalon-MM Bridge

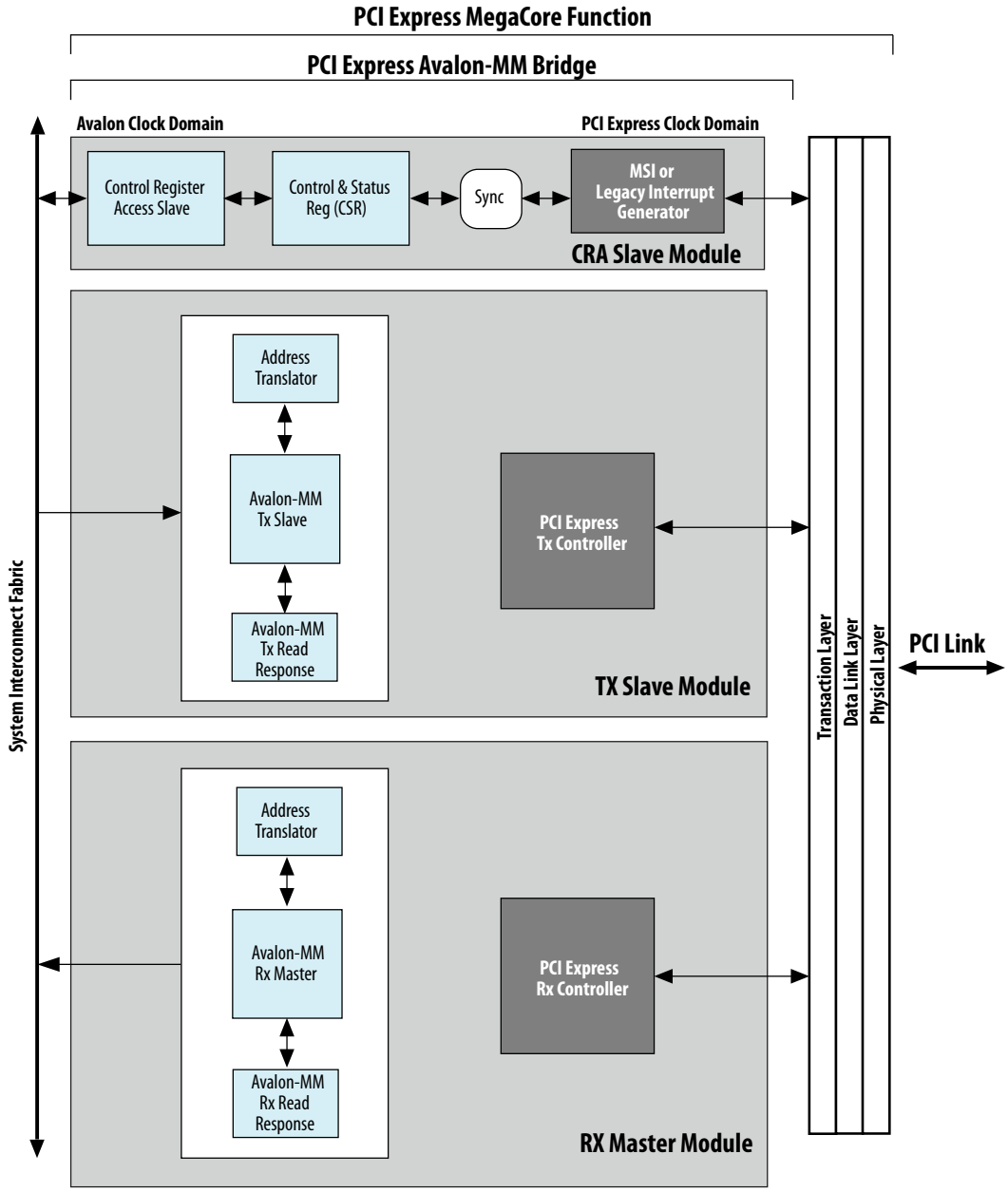
The Avalon-MM Arria 10 Hard IP for PCI Express includes an Avalon-MM bridge module that connects the Hard IP to the interconnect fabric. The bridge facilitates the design of Endpoints and Root Ports that include Qsys components.

The Avalon-MM bridge provides three possible Avalon-MM ports: a bursting master, an optional bursting slave, and an optional non-bursting slave. The Avalon-MM bridge comprises the following three modules:

- TX Slave Module—This optional 64- or 128-bit bursting, Avalon-MM dynamic addressing slave port propagates read and write requests of up to 4 KBytes in size from the interconnect fabric to the PCI Express link. The bridge translates requests from the interconnect fabric to PCI Express request packets.
- RX Master Module—This 64- or 128-bit bursting Avalon-MM master port propagates PCI Express requests, converting them to bursting read or write requests to the interconnect fabric.
- Control Register Access (CRA) Slave Module—This optional, 32-bit Avalon-MM dynamic addressing slave port provides access to internal control and status registers from upstream PCI Express devices and external Avalon-MM masters. Implementations that use MSI or dynamic address translation require this port. The CRA port supports single dword read and write requests. It does not support bursting.

When you select the **Single dword completer** for the Avalon-MM Hard IP for PCI Express, Qsys substitutes a unpipelined, 32-bit RX master port for the 64- or 128-bit full-featured RX master port. The following figure shows the block diagram of a full-featured PCI Express Avalon-MM bridge.

Figure 10-4: PCI Express Avalon-MM Bridge



The bridge has the following additional characteristics:

- Type 0 and Type 1 vendor-defined incoming messages are discarded
- Completion-to-a-flush request is generated, but not propagated to the interconnect fabric

For End Points, each PCI Express base address register (BAR) in the Transaction Layer maps to a specific, fixed Avalon-MM address range. You can use separate BARs to map to various Avalon-MM slaves connected to the RX Master port. In contrast to Endpoints, Root Ports do not perform any BAR matching and forwards the address to a single RX Avalon-MM master port.

Related Information

Avalon-MM RX Master Block on page 10-20

Avalon-MM Bridge TLPs

The PCI Express to Avalon-MM bridge translates the PCI Express read, write, and completion Transaction Layer Packets (TLPs) into standard Avalon-MM read and write commands typically used by master and slave interfaces. This PCI Express to Avalon-MM bridge also translates Avalon-MM read, write and read data commands to PCI Express read, write and completion TLPs. The following topics describe the Avalon-MM bridges translations.

Avalon-MM-to-PCI Express Write Requests

The Avalon-MM bridge accepts Avalon-MM burst write requests with a burst size of up to 512 Bytes at the Avalon-MM TX slave interface. The Avalon-MM bridge converts the write requests to one or more PCI Express write packets with 32– or 64-bit addresses based on the address translation configuration, the request address, and the maximum payload size.

The Avalon-MM write requests can start on any address in the range defined in the PCI Express address table parameters. The bridge splits incoming burst writes that cross a 4 KByte boundary into at least two separate PCI Express packets. The bridge also considers the root complex requirement for maximum payload on the PCI Express side by further segmenting the packets if needed.

The bridge requires Avalon-MM write requests with a burst count of greater than one to adhere to the following byte enable rules:

- The Avalon-MM byte enables must be asserted in the first qword of the burst.
- All subsequent byte enables must be asserted until the deasserting byte enable.
- The Avalon-MM byte enables may deassert, but only in the last qword of the burst.

Note: To improve PCI Express throughput, Altera recommends using an Avalon-MM burst master without any byte-enable restrictions.

Avalon-MM-to-PCI Express Upstream Read Requests

The PCI Express Avalon-MM bridge converts read requests from the system interconnect fabric to PCI Express read requests with 32-bit or 64-bit addresses based on the address translation configuration, the request address, and the maximum read size.

The Avalon-MM TX slave interface of a PCI Express Avalon-MM bridge can receive read requests with burst sizes of up to 512 bytes sent to any address. However, the bridge limits read requests sent to the PCI Express link to a maximum of 256 bytes. Additionally, the bridge must prevent each PCI Express read request packet from crossing a 4 KByte address boundary. Therefore, the bridge may split an Avalon-MM read request into multiple PCI Express read packets based on the address and the size of the read request.

Avalon-MM bridge supports up to eight outstanding reads from Avalon-MM interface. Once the bridge has eight outstanding read requests, the `txs_waitrequest` signal is asserted to block additional read requests. When a read request completes, the Avalon-MM bridge can accept another request.

For Avalon-MM read requests with a burst count greater than one, all byte enables must be asserted. There are no restrictions on byte enables for Avalon-MM read requests with a burst count of one. An invalid Avalon-MM request can adversely affect system functionality, resulting in a completion with the abort status set. An example of an invalid request is one with an incorrect address.

PCI Express-to-Avalon-MM Read Completions

The PCI Express Avalon-MM bridge returns read completion packets to the initiating Avalon-MM master in the issuing order. The bridge supports multiple and out-of-order completion packets.

PCI Express-to-Avalon-MM Downstream Write Requests

The PCI Express Avalon-MM bridge receives PCI Express write requests, it converts them to burst write requests before sending them to the interconnect fabric. For Endpoints, the bridge translates the PCI Express address to the Avalon-MM address space based on the BAR hit information and on address translation table values configured during the IP core parameterization. For Root Ports, all requests are forwarded to a single RX Avalon-MM master that drives them to the interconnect fabric. Malformed write packets are dropped, and therefore do not appear on the Avalon-MM interface.

For downstream write and read requests, if more than one byte enable is asserted, the byte lanes must be adjacent. In addition, the byte enables must be aligned to the size of the read or write request.

As an example, the following table lists the byte enables for 32-bit data.

Table 10-2: Valid Byte Enable Configurations

Byte Enable Value	Description
4'b1111	Write full 32 bits
4'b0011	Write the lower 2 bytes
4'b1100	Write the upper 2 bytes
4'b0001	Write byte 0 only
4'b0010	Write byte 1 only
4'b0100	Write byte 2 only
4'b1000	Write byte 3 only

In burst mode, the Arria 10 Hard IP for PCI Express supports only byte enable values that correspond to a contiguous data burst. For the 32-bit data width example, valid values in the first data phase are 4'b1111, 4'b1110, 4'b1100, and 4'b1000, and valid values in the final data phase of the burst are 4'b1111, 4'b0111, 4'b0011, and 4'b0001. Intermediate data phases in the burst can only have byte enable value 4'b1111.

PCI Express-to-Avalon-MM Downstream Read Requests

The PCI Express Avalon-MM bridge sends PCI Express read packets to the interconnect fabric as burst reads with a maximum burst size of 512 bytes. For Endpoints, the bridge converts the PCI Express address to the Avalon-MM address space based on the BAR hit information and address translation lookup table values. The RX Avalon-MM master port drives the received address to the fabric. You can set up the Address Translation Table Configuration in the parameter editor. Unsupported read requests generate a completer abort response.

Related Information

Minimizing BAR Sizes and the PCIe Address Space on page 10-15

Avalon-MM-to-PCI Express Read Completions

The PCI Express Avalon-MM bridge converts read response data from Application Layer Avalon-MM slaves to PCI Express completion packets and sends them to the Transaction Layer.

A single read request may produce multiple completion packets based on the **Maximum payload size** and the size of the received read request. For example, if the read is 512 bytes but the **Maximum payload size** 128 bytes, the bridge produces four completion packets of 128 bytes each. The bridge does not generate out-of-order completions even to different BARs. You can specify the **Maximum payload size** parameter on the **Device** tab under the **PCI Express/PCI Capabilities** heading in the parameter editor.

Related Information

Device Capabilities

PCI Express-to-Avalon-MM Address Translation for 32-Bit Bridge

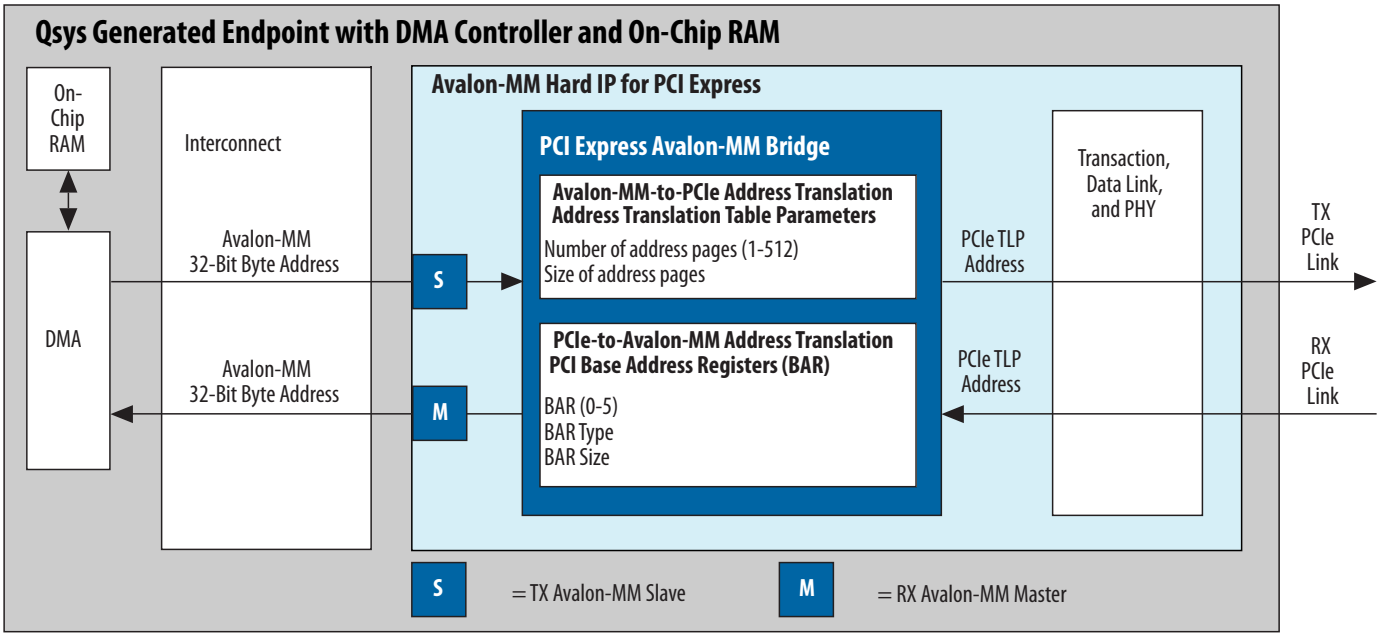
The PCI Express Avalon-MM bridge translates the system-level physical addresses, typically up to 64 bits, to the significantly smaller addresses required by the Application Layer's Avalon-MM slave components.

Note: Starting with the 13.0 version of the Quartus II software, the PCI Express-to-Avalon-MM bridge supports both 32- and 64-bit addresses. If you select 64-bit addressing the bridge does not perform address translation. It drives the addresses specified to the interconnect fabric. You can limit the number of address bits used by Avalon-MM slave components to the actual size required by specifying the address size in the Avalon-MM slave component parameter editor.

You can specify up to six BARs for address translation when you customize your Hard IP for PCI Express as described in *Base Address Register (BAR) and Expansion ROM Settings*. When 32-bit addresses are specified, the PCI Express Avalon-MM bridge also translates Application Layer addresses to system-level physical addresses as described in *Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing*.

The following figure provides a high-level view of address translation in both directions.

Figure 10-5: Address Translation in TX and RX Directions For Endpoints



Note: When configured as a Root Port, a single RX Avalon-MM master forwards all RX TLPs to the Qsys interconnect.

The Avalon-MM RX master module port has an 8-byte datapath in 64-bit mode and a 16-byte datapath in 128-bit mode. The Qsys interconnect fabric manages mismatched port widths transparently.

As Memory Request TLPs are received from the PCIe link, the most significant bits are used in the BAR matching as described in the PCI specifications. The least significant bits not used in the BAR match process are passed unchanged as the Avalon-MM address for that BAR's RX Master port.

For example, consider the following configuration specified using the Base Address Registers in the parameter editor:

1. BAR1:0 is a **64-bit prefetchable memory** that is **4KBytes -12 bits**
2. System software programs BAR1:0 to have a base address of 0x0000123456789000
3. A TLP received with address 0x0000123456789870
4. The upper 52 bits (0x0000123456789) are used in the BAR matching process, so this request matches.
5. The lower 12 bits, 0x870, are passed through as the Avalon address on the Rxm_BAR0 Avalon-MM Master port. The BAR matching software replaces the upper 20 bits of the address with the Avalon-MM base address.

Related Information

Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing on page 10-17

Minimizing BAR Sizes and the PCIe Address Space

For designs that include multiple BARs, you may need to modify the base address assignments auto-assigned by Qsys in order to minimize the address space that the BARs consume. For example, consider a Qsys system with the following components:

- **Offchip_Data_Mem DDR3** (SDRAM Controller with UniPHY) controlling 256 MBytes of memory—Qsys auto-assigned a base address of 0x00000000
- **Quick_Data_Mem** (On-Chip Memory (RAM or ROM)) of 4 KBytes—Qsys auto-assigned a base address of 0x10000000
- **Instruction_Mem** (On-Chip Memory (RAM or ROM)) of 64 KBytes—Qsys auto-assigned a base address of 0x10020000
- **PCIe** (Avalon-MM Arria 10 Hard IP for PCI Express)
  - **Cra** (Avalon-MM Slave)—auto assigned base address of 0x10004000
  - **Rxm_BAR0** connects to **Offchip_Data_Mem DD R3 avl**
  - **Rxm_BAR2** connects to **Quick_Data_Mem s1**
  - **Rxm_BAR4** connects to PCIe. **Cra Avalon MM Slave**
- **Nios2** (Nios® II Processor)
  - **data_master** connects to **PCIe Cra**, **Offchip_Data_Mem DDR3 avl**, **Quick_Data_Mem s1**, **Instruction_Mem s1**, **Nios2 jtag_debug_module**
  - **instruction_master** connects to **Instruction_Mem s1**

Figure 10-6: Qsys System for PCI Express with Poor Address Space Utilization

The following figure uses a filter to hide the Conduit interfaces that are not relevant in this discussion.

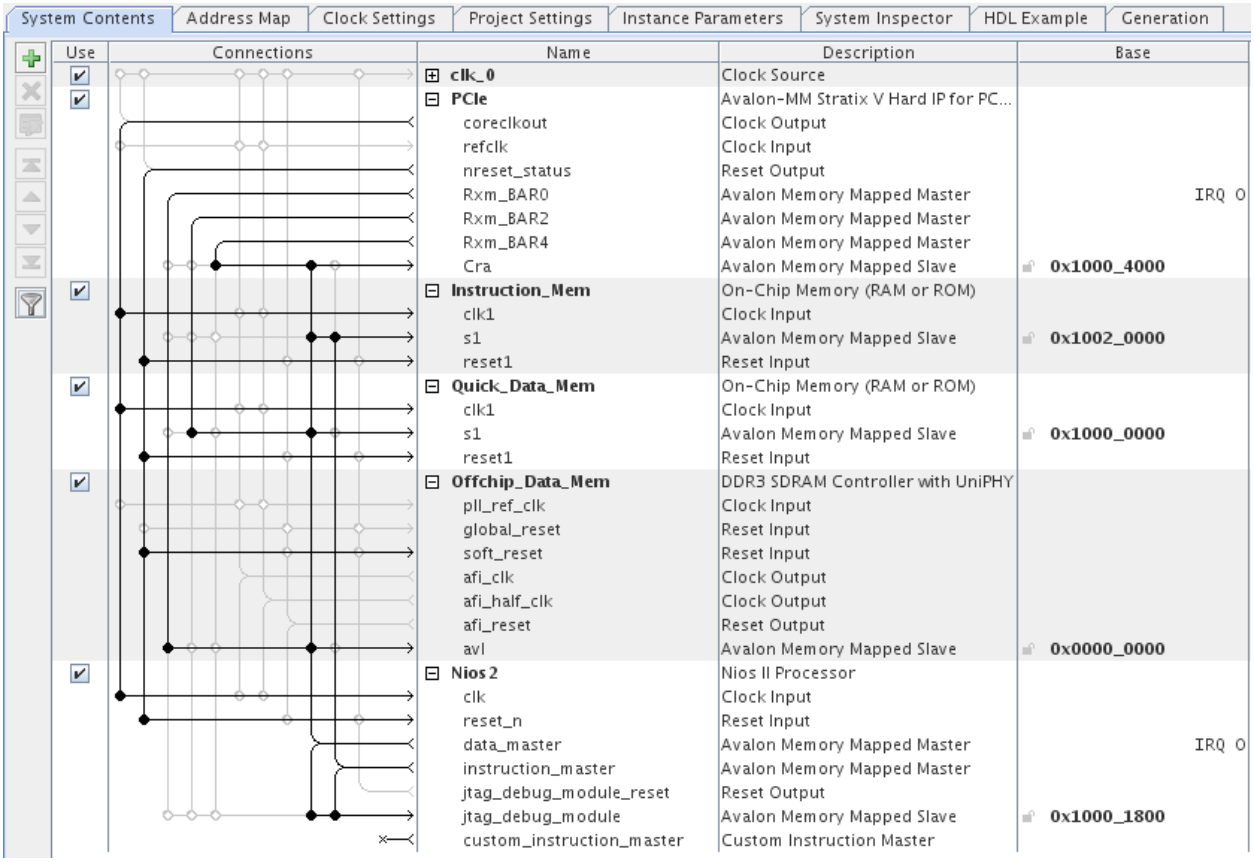
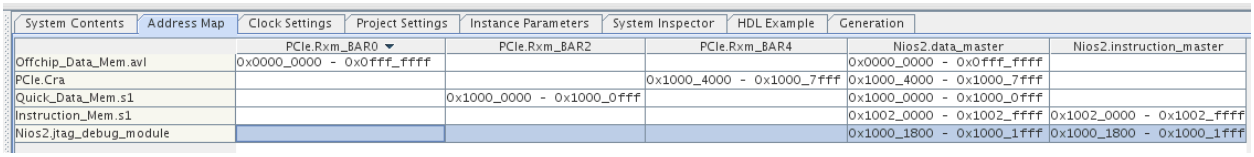


Figure 10-7: Poor Address Map

The following figure illustrates the address map for this system.



	PCIe.Rxm_BAR0	PCIe.Rxm_BAR2	PCIe.Rxm_BAR4	Nios2.data_master	Nios2.instruction_master
Offchip_Data_Mem.avl	0x0000_0000 - 0x0fff_ffff			0x0000_0000 - 0x0fff_ffff	
PCIe.Cra			0x1000_4000 - 0x1000_7fff	0x1000_4000 - 0x1000_7fff	
Quick_Data_Mem.s1		0x1000_0000 - 0x1000_0fff		0x1000_0000 - 0x1000_0fff	
Instruction_Mem.s1				0x1002_0000 - 0x1002_ffff	0x1002_0000 - 0x1002_ffff
Nios2.jtag_debug_module				0x1000_1800 - 0x1000_1fff	0x1000_1800 - 0x1000_1fff

The auto-assigned base addresses result in the following three large BARs:

- BAR0 is 28 bits. This is the optimal size because it addresses the **Offchip_Data_Mem** which requires 28 address bits.
- BAR2 is 29 bits. BAR2 addresses the **Quick_Data_Mem** which is 4 KBytes;. It should only require 12 address bits; however, it is consuming 512 MBytes of address space.
- BAR4 is also 29 bits. BAR4 address **PCIe Cra** is 16 KBytes. It should only require 14 address bits; however, it is also consuming 512 MBytes of address space.

This design is consuming 1.25 GB of PCIe address space when only 276 MBytes are actually required. The solution is to edit the address map to place the base address of each BAR at 0x0000_0000. The following figure illustrates the optimized address map.

Figure 10-8: Optimized Address Map

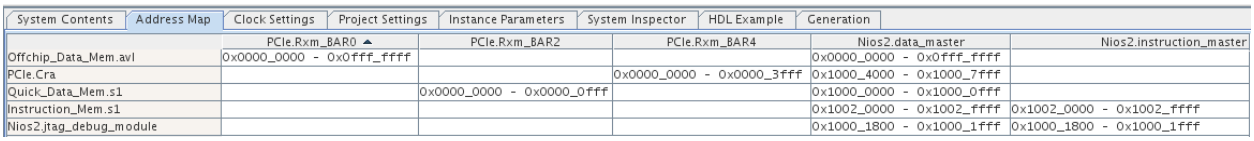
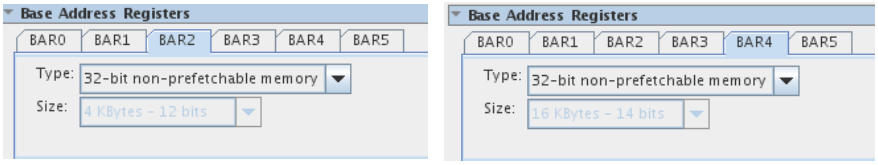


Figure 10-9: Reduced Address Bits for BAR2 and BAR4

The following figure shows the number of address bits required when the smaller memories accessed by BAR2 and BAR4 have a base address of 0x0000_0000.



For cases where the BAR Avalon-MM RX master port connects to more than one Avalon-MM slave, assign the base addresses of the slaves sequentially and place the slaves in the smallest power-of-two-sized address space possible. Doing so minimizes the system address space used by the BAR.

Related Information

Address Map Tab (Qsys)

Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing

Note: The PCI Express-to-Avalon-MM bridge supports both 32- and 64-bit addresses. If you select 64-bit addressing the bridge does not perform address translation.

When you specify 32-bit addresses, the Avalon-MM address of a received request on the TX Avalon-MM slave port is translated to the PCI Express address before the request packet is sent to the Transaction Layer. You can specify up to 512 address pages and sizes ranging from 4 KByte to 4 GBytes when you customize your Avalon-MM Arria 10 Hard IP for PCI Express as described in *Avalon to PCIe Address Translation Settings* . This address translation process proceeds by replacing the MSB of the Avalon-MM address with the value from a specific translation table entry; the LSB remains unchanged. The number of MSBs to be replaced is calculated based on the total address space of the upstream PCI Express devices that the Avalon-MM Hard IP for PCI Express can access. The number of MSB bits is defined by the difference between the maximum number of bits required to represent the address space supported by the upstream PCI Express device minus the number of bits required to represent the **Size of address pages** which are the LSB pass-through bits (*N*). The **Size of address pages** (*N*) is applied to all entries in the translation table.

Each of the 512 possible entries corresponds to the base address of a PCI Express memory segment of a specific size. The segment size of each entry must be identical. The total size of all the memory segments is used to determine the number of address MSB to be replaced. In addition, each entry has a 2-bit field,

`Sp[1:0]`, that specifies 32-bit or 64-bit PCI Express addressing for the translated address. The most significant bits of the Avalon-MM address are used by the interconnect fabric to select the slave port and are not available to the slave. The next most significant bits of the Avalon-MM address index the address translation entry to be used for the translation process of MSB replacement.

For example, if the core is configured with an address translation table with the following attributes:

- **Number of Address Pages—16**
- **Size of Address Pages—1 MByte**
- **PCI Express Address Size—64 bits**

then the values in the following figure are:

- $N = 20$ (due to the 1 MByte page size)
- $Q = 16$ (number of pages)
- $M = 24$ (20 + 4 bit page selection)
- $P = 64$

In this case, the Avalon address is interpreted as follows:

- Bits [31:24] select the TX slave module port from among other slaves connected to the same master by the system interconnect fabric. The decode is based on the base addresses assigned in Qsys.
- Bits [23:20] select the address translation table entry.
- Bits [63:20] of the address translation table entry become PCI Express address bits [63:20].
- Bits [19:0] are passed through and become PCI Express address bits [19:0].

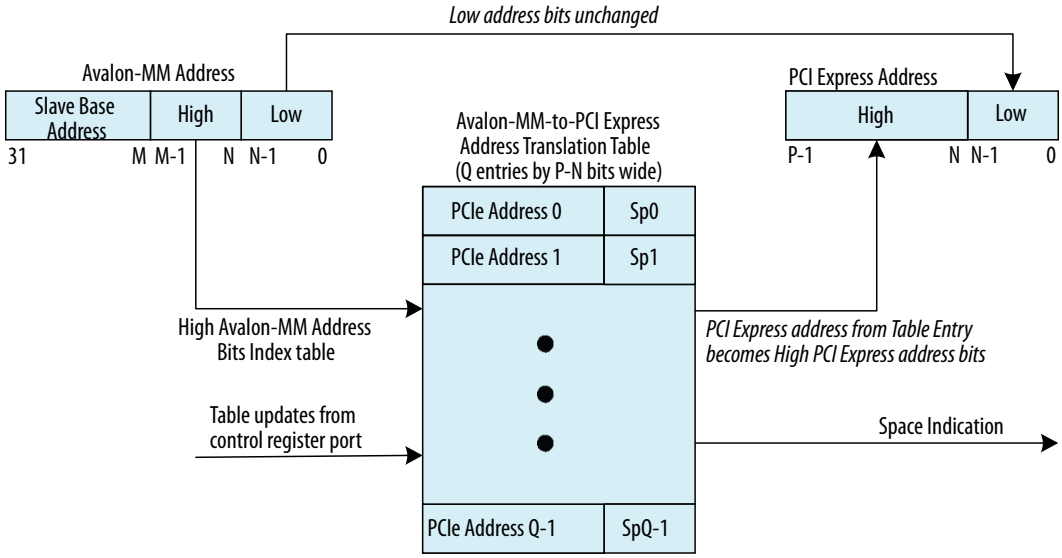
The address translation table is dynamically configured at run time. The address translation table is implemented in memory and can be accessed through the CRA slave module. Dynamic configuration is optimal in a typical PCI Express system where address allocation occurs after BIOS initialization.

For more information about how to access the dynamic address translation table through the CRA slave, refer to the “Avalon-MM-to-PCI Express Address Translation Table 0x1000–0x1FFF” on page 9–17.

Figure 10-10: Avalon-MM-to-PCI Express Address Translation

The following figure depicts the Avalon-MM-to-PCI Express address translation process. In this figure the variables represent the following paramers:

- *N*—the number of pass-through bits.
- *M*—the number of Avalon-MM address bits.
- *P*—the number of PCIe address bits.
- *Q*—the number of translation table entries.
- `Sp[1:0]`—the space indication for each entry.



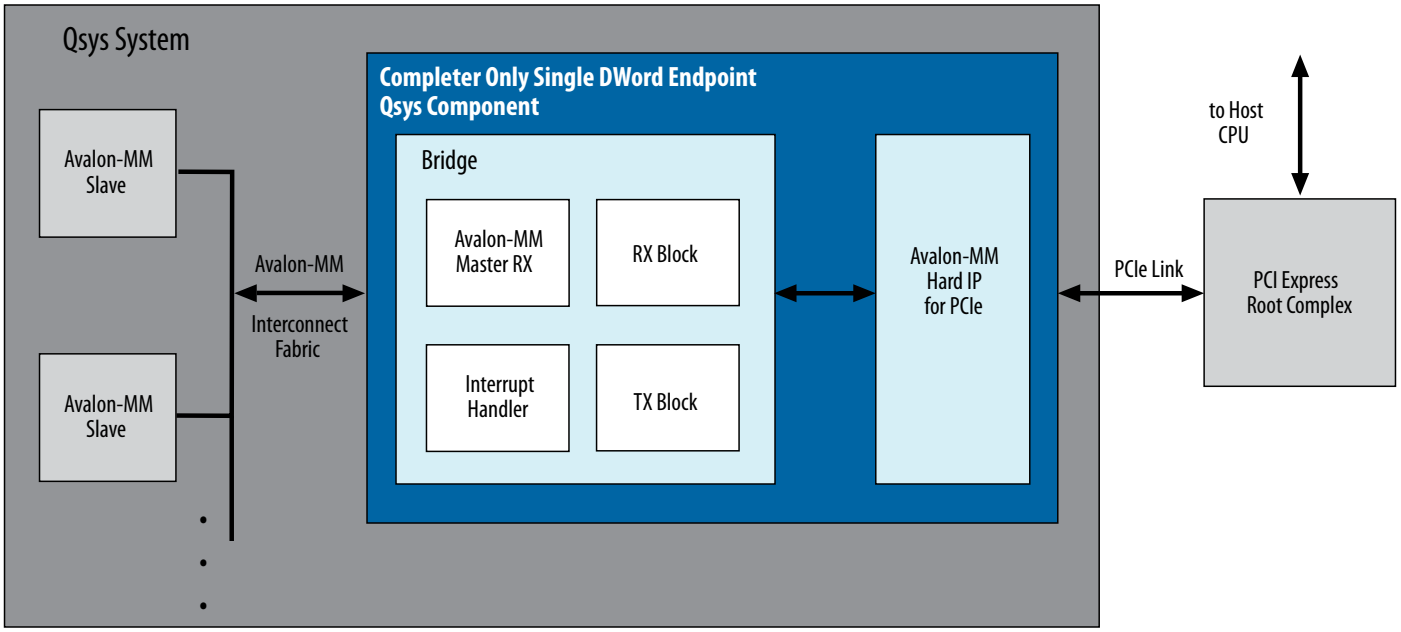
Completer Only Single Dword Endpoint

The completer only single dword endpoint is intended for applications that use the PCI Express protocol to perform simple read and write register accesses from a host CPU. The completer only single dword endpoint is a hard IP implementation available for Qsys systems, and includes an Avalon-MM interface to the Application Layer. The Avalon-MM interface connection in this variation is 32 bits wide. This endpoint is not pipelined; at any time a single request can be outstanding.

The completer-only single dword endpoint supports the following requests:

- Read and write requests of a single dword (32 bits) from the Root Complex
- Completion with Completer Abort status generation for other types of non-posted requests
- INTX or MSI support with one Avalon-MM interrupt source

Figure 10-11: Qsys Design Including Completer Only Single Dword Endpoint for PCI Express



The above figure shows the that completer-only single dword endpoint connects to a PCI Express root complex. A bridge component includes the Arria 10 Hard IP for PCI Express TX and RX blocks, an Avalon-MM RX master, and an interrupt handler. The bridge connects to the FPGA fabric using an Avalon-MM interface. The following sections provide an overview of each block in the bridge.

RX Block

The RX Block control logic interfaces to the hard IP block to process requests from the root complex. It supports memory reads and writes of a single dword. It generates a completion with Completer Abort (CA) status for read requests greater than four bytes and discards all write data without further action for write requests greater than four bytes.

The RX block passes header information to the Avalon-MM master, which generates the corresponding transaction to the Avalon-MM interface. The bridge accepts no additional requests while a request is being processed. While processing a read request, the RX block deasserts the `ready` signal until the TX block sends the corresponding completion packet to the hard IP block. While processing a write request, the RX block sends the request to the Avalon-MM interconnect fabric before accepting the next request.

Avalon-MM RX Master Block

The 32-bit Avalon-MM master connects to the Avalon-MM interconnect fabric. It drives read and write requests to the connected Avalon-MM slaves, performing the required address translation. The RX master supports all legal combinations of byte enables for both read and write requests.

For more information about legal combinations of byte enables, refer to *Avalon Memory Mapped Interfaces* in the Avalon Interface Specifications.

Related Information

Avalon Interface Specifications

TX Block

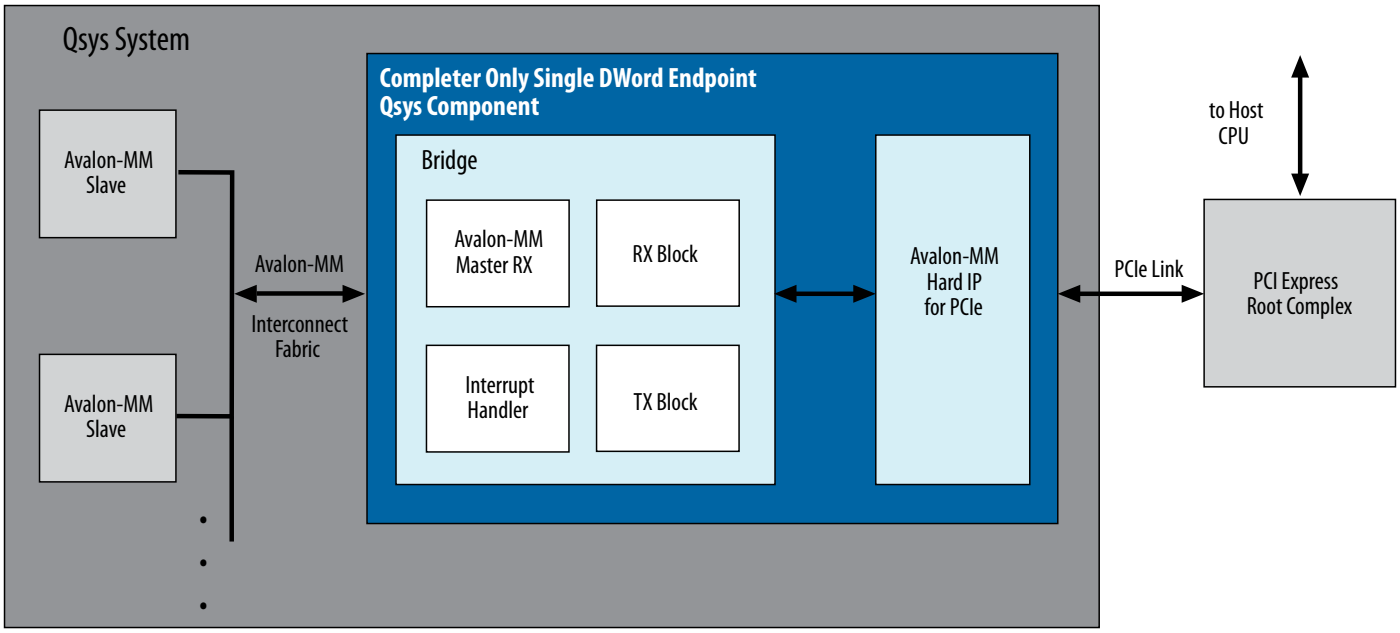
The TX block sends completion information to the Avalon-MM Hard IP for PCI Express which sends this information to the root complex. The TX completion block generates a completion packet with Completer Abort (CA) status and no completion data for unsupported requests. The TX completion block also supports the zero-length read (flush) command.

Interrupt Handler Block

The interrupt handler implements both INTX and MSI interrupts. The `msi_enable` bit in the configuration register specifies the interrupt type. The `msi_enable_bit` is part of the MSI message control portion in the MSI Capability structure. It is bit[16] of address 0x050 in the Configuration Space registers. If the `msi_en able` bit is on, an MSI request is sent to the Arria 10 Hard IP for PCI Express when received, otherwise INTX is signaled. The interrupt handler block supports a single interrupt source, so that software may assume the source. You can disable interrupts by leaving the interrupt signal unconnected in the IRQ column of Qsys.

When the MSI registers in the Configuration Space of the Completer Only Single Dword Arria 10 Hard IP for PCI Express are updated, there is a delay before this information is propagated to the Bridge module shown in the following figure.

Figure 10-12: Qsys Design Including Completer Only Single Dword Endpoint for PCI Express



You must allow time for the Bridge module to update the MSI register information. Normally, setting up MSI registers occurs during enumeration process. Under normal operation, initialization of the MSI registers should occur substantially before any interrupt is generated. However, failure to wait until the update completes may result in any of the following behaviors:

- Sending a legacy interrupt instead of an MSI interrupt
- Sending an MSI interrupt instead of a legacy interrupt
- Loss of an interrupt request

According to the *PCI Express Base Specification*, if `MSI_enable`=0 and the `Disable Legacy Interrupt bit`=1 in the Configuration Space `Command` register (0x004), the Hard IP should not send legacy interrupt messages when an interrupt is generated.

Design Implementation 11

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

Completing your design includes additional steps to specify analog properties, pin assignments, and timing constraints.

Making Pin Assignments to Assign I/O Standard to Serial Data Pins

Before running Quartus II compilation, use the **Pin Planner** to assign I/O standards to the pins of the device.

1. On the Quartus II **Assignments** menu, select **Pin Planner**.
   The **Pin Planner** appears.
2. In the **Node Name** column, locate the PCIe serial data pins.
3. In the **I/O Standard** column, double-click the right-hand corner of the box to bring up a list of available I/O standards.
4. Select the appropriate standard from the following table.

Table 11-1: I/O Standards for HSSI Pins

Pin Type	I/O Standard
HSSI REFCLK	**Current Mode Logic (CML), HCSL**
HSSI RX	**Current Mode Logic (CML)**
HSSI TX	**High Speed Differential I/O**

The Quartus II software adds instance assignments to your Quartus II Settings File (***.qsf**). The assignment is in the form `set_instance_assignment -name IO_STANDARD <"IO_STANDARD_NAME"> -to <signal_name>`. The ***.qsf** is in your synthesis directory.

Related Information

Arria 10 GX GT, and SX Device Family Pin Connection Guidelines

Recommended Reset Sequence to Avoid Link Training Issues

Successful link training can only occur after the FPGA is configured. Designs using CvP for configuration initially load the I/O ring and periphery image. Arria 10 devices include a Nios II Hard Calibration IP core that automatically calibrates transceivers to optimize signal quality after CvP completes and before

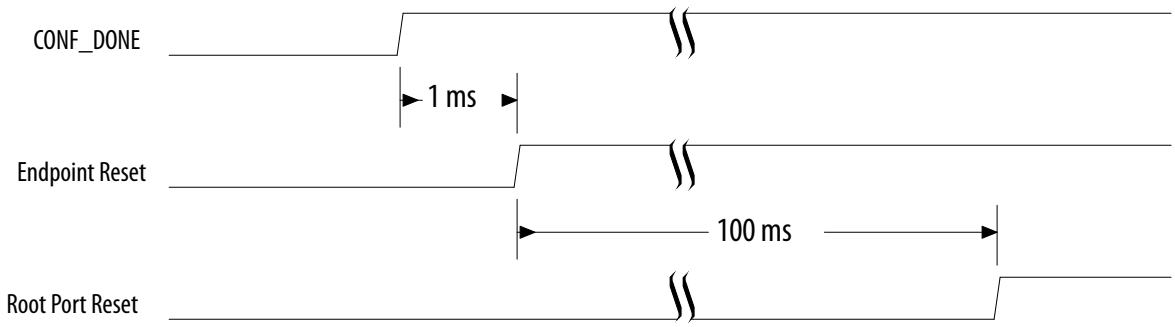
© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

entering user mode. Link training occurs after calibration. Refer to *Reset Sequence for Hard IP for PCI Express IP Core and Application Layer* for a description of the key signals that reset, control dynamic reconfiguration, and link training. Altera recommends separate control of reset signals for the Endpoint and Root Port. Successful reset sequence includes the following steps:

1. Wait until the FPGA is configured as indicated by the assertion of `CONFIG_DONE` from the FPGA block controller.
2. Wait 1 ms after the assertion of `CONFIG_DONE`, then deassert the Endpoint reset.
3. Wait approximately 100 ms, then deassert the Root Port reset.
4. Deassert the reset output to the Application Layer.

Figure 11-1: Recommended Reset Sequence



Related Information

Arria 10 Transceiver PHY User Guide
For information about requirements for the `CLKUSR` pin used during automatic calibration.

SDC Timing Constraints

Your top-level Synopsys Design Constraints file (**.sdc**) must include the following timing constraint macro for the Arria 10 Hard IP for PCIe IP core.

Example 11-1: SDC Timing Constraints Required for the Arria 10 Hard IP for PCIe and Design Example

```
# Constraints required for the Arria 10 Hard IP for PCI Express
# derive_pll_clock is used to calculate all clock derived
# from PCIe refclk. It must be applied once across all
# of the SDC files used in a project
derive_pll_clocks -create_base_clocks
```

You should only include this constraint in one location across all of the SDC files in your project. Differences between Fitter timing analysis and TimeQuest timing analysis arise if these constraints are applied multiple times.

Throughput Optimization 12

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

The *PCI Express Base Specification* defines a flow control mechanism to ensure efficient transfer of TLPs.

Each transmitter, the write requester in this case, maintains a `credit limit` register and a `credits consumed` register. The `credit limit` register is the sum of all credits received by the receiver, the write completer in this case. The `credit limit` register is initialized during the flow control initialization phase of link initialization and then updated during operation by Flow Control (FC) Update DLLPs. The `credits consumed` register is the sum of all credits consumed by packets transmitted. Separate `credit limit` and `credits consumed` registers exist for each of the six types of Flow Control:

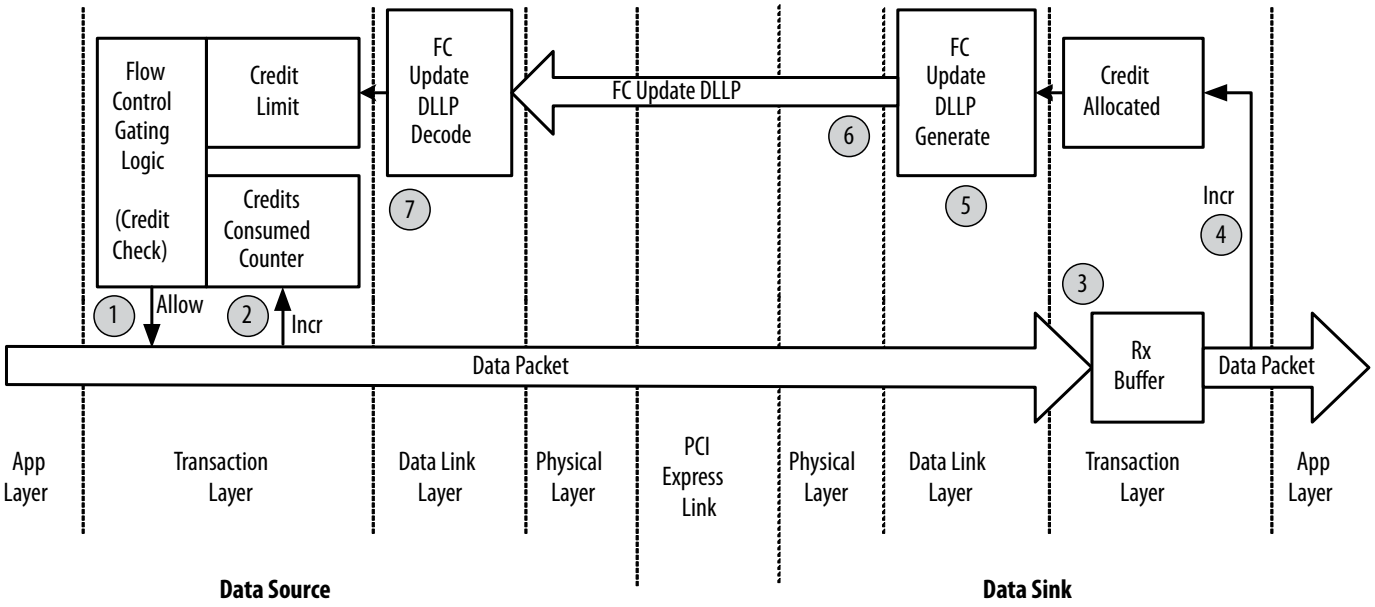
- Posted Headers
- Posted Data
- Non-Posted Headers
- Non-Posted Data
- Completion Headers
- Completion Data

Each receiver also maintains a `credit allocated` counter which is initialized to the total available space in the RX buffer (for the specific Flow Control class) and then incremented as packets are pulled out of the RX buffer by the Application Layer. The value of this register is sent as the FC Update DLLP value.

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure 12-1: Flow Control Update Loop



The following numbered steps describe each step in the Flow Control Update loop. The corresponding numbers in the figure show the general area to which they correspond.

1. When the Application Layer has a packet to transmit, the number of credits required is calculated. If the current value of the credit limit minus credits consumed is greater than or equal to the required credits, then the packet can be transmitted immediately. However, if the credit limit minus credits consumed is less than the required credits, then the packet must be held until the credit limit is increased to a sufficient value by an FC Update DLLP. This check is performed separately for the header and data credits; a single packet consumes only a single header credit.
2. After the packet is selected for transmission the `credits consumed` register is incremented by the number of credits consumed by this packet. This increment happens for both the header and data `credit consumed` registers.
3. The packet is received at the other end of the link and placed in the RX buffer.
4. At some point the packet is read out of the RX buffer by the Application Layer. After the entire packet is read out of the RX buffer, the `credit allocated` register can be incremented by the number of credits the packet has used. There are separate `credit allocated` registers for the header and data credits.
5. The value in the `credit allocated` register is used to create an FC Update DLLP.
6. After an FC Update DLLP is created, it arbitrates for access to the PCI Express link. The FC Update DLLPs are typically scheduled with a low priority; consequently, a continuous stream of Application Layer TLPs or other DLLPs (such as ACKs) can delay the FC Update DLLP for a long time. To prevent starving the attached transmitter, FC Update DLLPs are raised to a high priority under the following three circumstances:
   a. When the last sent `credit allocated` counter minus the amount of received data is less than `MAX_PAYLOAD` and the current `credit allocated` counter is greater than the last sent credit

counter. Essentially, this means the data sink knows the data source has less than a full `MAX_PAYLOAD` worth of credits, and therefore is starving.

b. When an internal timer expires from the time the last FC Update DLLP was sent, which is configured to 30 µs to meet the *PCI Express Base Specification* for resending FC Update DLLPs.

c. When the `credit allocated` counter minus the last sent `credit allocated` counter is greater than or equal to 25% of the total credits available in the RX buffer, then the FC Update DLLP request is raised to high priority.

After arbitrating, the FC Update DLLP that won the arbitration to be the next item is transmitted. In the worst case, the FC Update DLLP may need to wait for a maximum sized TLP that is currently being transmitted to complete before it can be sent.

7. The original write requester receives the FC Update DLLP. The `credit limit` value is updated. If packets are stalled waiting for credits, they can now be transmitted.

Note: You must keep track of the credits consumed by the Application Layer.

Throughput of Posted Writes

The throughput of posted writes is limited primarily by the Flow Control Update loop as shown in **Figure 12-1**. If the write requester sources the data as quickly as possible, and the completer consumes the data as quickly as possible, then the Flow Control Update loop may be the biggest determining factor in write throughput, after the actual bandwidth of the link.

The figure below shows the main components of the Flow Control Update loop with two communicating PCI Express ports:

- Write Requester
- Write Completer

To allow the write requester to transmit packets continuously, the `credit allocated` and the `credit limit` counters must be initialized with sufficient credits to allow multiple TLPs to be transmitted while waiting for the FC Update DLLP that corresponds to the freeing of credits from the very first TLP transmitted.

You can use the **RX Buffer space allocation - Desired performance for received requests** to configure the RX buffer with enough space to meet the credit requirements of your system.

Related Information

PCI Express Base Specification 3.0

Throughput of Non-Posted Reads

To support a high throughput for read data, you must analyze the overall delay from the time the Application Layer issues the read request until all of the completion data is returned. The Application Layer must be able to issue enough read requests, and the read completer must be capable of processing these read requests quickly enough (or at least offering enough non-posted header credits) to cover this delay.

However, much of the delay encountered in this loop is well outside the IP core and is very difficult to estimate. PCI Express switches can be inserted in this loop, which makes determining a bound on the delay more difficult.

Nevertheless, maintaining maximum throughput of completion data packets is important. Endpoints must offer an infinite number of completion credits. Endpoints must buffer this data in the RX buffer until the Application Layer can process it. Because the Endpoint is no longer managing the RX buffer for Completions through the flow control mechanism, the Application Layer must manage the RX buffer by the rate at which it issues read requests.

To determine the appropriate settings for the amount of space to reserve for completions in the RX buffer, you must make an assumption about the length of time until read completions are returned. This assumption can be estimated in terms of an additional delay, beyond the FC Update Loop Delay, as discussed in the section *Throughput of Posted Writes*. The paths for the read requests and the completions are not exactly the same as those for the posted writes and FC Updates in the PCI Express logic. However, the delay differences are probably small compared with the inaccuracy in the estimate of the external read to completion delays.

With multiple completions, the number of available credits for completion headers must be larger than the completion data space divided by the maximum packet size. Instead, the credit space for headers must be the completion data space (in bytes) divided by 64, because this is the smallest possible read completion boundary. Setting the **RX Buffer space allocation—Desired performance for received completions** to **High** under the **System Settings** heading when specifying parameter settings configures the RX buffer with enough space to meet this requirement. You can adjust this setting up or down from the **High** setting to tailor the RX buffer size to your delays and required performance.

Optional Features 13

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

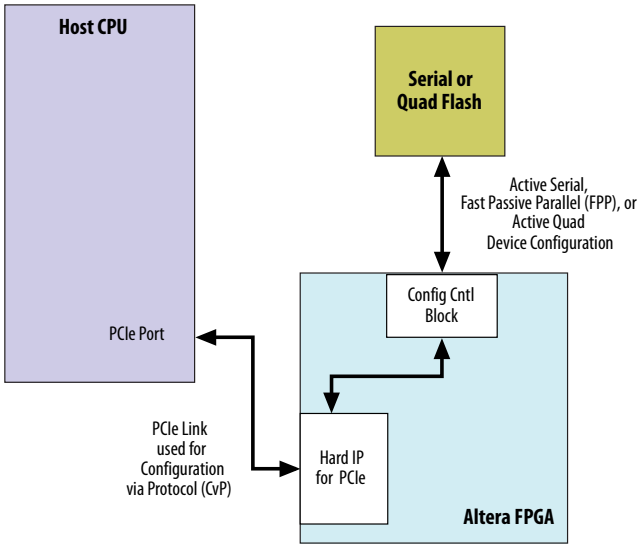
Configuration via Protocol (CvP)

The Hard IP for PCI Express architecture has an option to configure the FPGA and initialize the PCI Express link. In prior devices, a single Program Object File (**.pof**) programmed the I/O ring and FPGA fabric before the PCIe link training and enumeration began. The **.pof** file is divided into two parts:

- The I/O bitstream contains the data to program the I/O ring, the Hard IP for PCI Express, and other elements that are considered part of the periphery image.
- The core bitstream contains the data to program the FPGA fabric.

When you select the CvP design flow, the I/O ring and PCI Express link are programmed first, allowing the PCI Express link to reach the L0 state and begin operation independently, before the rest of the core is programmed. After the PCI Express link is established, it can be used to program the rest of the device. The following figure shows the blocks that implement CvP.

Figure 13-1: CvP in Arria 10 Devices


Host CPU
Serial or Quad Flash
Active Serial, Fast Passive Parallel (FPP), or Active Quad Device Configuration
Config Cntl Block
PCIe Port
PCIe Link used for Configuration via Protocol (CvP)
Hard IP for PCIe
Altera FPGA

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

CvP has the following advantages:

- Provides a simpler software model for configuration. A smart host can use the PCIe protocol and the application topology to initialize and update the FPGA fabric.
- Enables dynamic core updates without requiring a system power down.
- Improves security for the proprietary core bitstream.
- Reduces system costs by reducing the size of the flash device to store the **.pof**.
- Facilitates hardware acceleration.
- May reduce system size because a single CvP link can be used to configure multiple FPGAs.

Related Information

Configuration via Protocol (CvP) Implementation in Altera FPGAs User Guide

ECRC

ECRC ensures end-to-end data integrity for systems that require high reliability. You can specify this option under the **Error Reporting** heading. The ECRC function includes the ability to check and generate ECRC. In addition, the ECRC function can forward the TLP with ECRC to the RX port of the Application Layer. When using ECRC forwarding mode, the ECRC check and generation are performed in the Application Layer.

You must turn on **Advanced error reporting (AER)**, **ECRC checking**, and **ECRC generation** under the **PCI Express/PCI Capabilities** heading using the parameter editor to enable this functionality.

For more information about error handling, refer to *Error Signaling and Logging* in Section 6.2 of the *PCI Express Base Specification*.

Related Information

PCI Express Base Specification 3.0

ECRC on the RX Path

When the **ECRC generation** option is turned on, errors are detected when receiving TLPs with a bad ECRC. If the **ECRC generation** option is turned off, no error detection occurs. If the **ECRC forwarding** option is turned on, the ECRC value is forwarded to the Application Layer with the TLP. If the **ECRC forwarding** option is turned off, the ECRC value is not forwarded.

Table 13-1: ECRC Operation on RX Path

ECRC Forwarding	ECRC Check Enable (4)	ECRC Status	Error	TLP Forward to Application Layer
No	No	none	No	Forwarded
		good	No	Forwarded without its ECRC
		bad	No	Forwarded without its ECRC
	Yes	none	No	Forwarded
		good	No	Forwarded without its ECRC
		bad	Yes	Not forwarded
Yes	No	none	No	Forwarded
		good	No	Forwarded with its ECRC
		bad	No	Forwarded with its ECRC
	Yes	none	No	Forwarded
		good	No	Forwarded with its ECRC
		bad	Yes	Not forwarded

ECRC on the TX Path

When the **ECRC generation** option is on, the TX path generates ECRC. If you turn on **ECRC forwarding**, the ECRC value is forwarded with the TLP. The following table summarizes the TX ECRC generation and forwarding. All unspecified cases are unsupported and the behavior of the Hard IP is unknown.In this table, if `TD` is 1, the TLP includes an ECRC. `TD` is the TL digest bit of the TL packet.

(4) The `ECRC Check Enable` field is in the `Configuration Space Advanced Error Capabilities and Control Register`.

Table 13-2: ECRC Generation and Forwarding on TX Path

All unspecified cases are unsupported and the behavior of the Hard IP is unknown.

ECRC Forwarding	ECRC Generation Enable (5)	TLP on Application	TLP on Link	Comments
No	No	TD=0, without ECRC	TD=0, without ECRC	
		TD=1, without ECRC	TD=0, without ECRC	
	Yes	TD=0, without ECRC	TD=1, with ECRC	ECRC is generated
		TD=1, without ECRC	TD=1, with ECRC	
Yes	No	TD=0, without ECRC	TD=0, without ECRC	Core forwards the ECRC
		TD=1, with ECRC	TD=1, with ECRC	
	Yes	TD=0, without ECRC	TD=0, without ECRC	
		TD=1, with ECRC	TD=1, with ECRC	

(5) The ECRC Generation Enable field is in the Configuration Space Advanced Error Capabilities and Control Register.

Avalon-MM Testbench and Design Example 14

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

This chapter introduces the Root Port or Endpoint design example including a testbench, BFM, and a test driver module. You can create this design example for using design flows described in *Getting Started with the Arria 10 Hard IP for PCI Express* .

When configured as an Endpoint variation, the testbench instantiates a design example and a Root Port BFM, which provides the following functions:

- A configuration routine that sets up all the basic configuration registers in the Endpoint. This configuration allows the Endpoint application to be the target and initiator of PCI Express transactions.
- A Verilog HDL procedure interface to initiate PCI Express transactions to the Endpoint.

The testbench uses a test driver module, **altera_avalon_dma** to exercise the DMA of the design example. The test driver module displays information from the Endpoint Configuration Space registers, so that you can correlate to the parameters you specified using the parameter editor.

When configured as a Root Port, the testbench instantiates a Root Port design example and an Endpoint model, which provides the following functions:

- A configuration routine that sets up all the basic configuration registers in the Root Port and the Endpoint BFM. This configuration allows the Endpoint application to be the target and initiator of PCI Express transactions.
- A Verilog HDL procedure interface to initiate PCI Express transactions to the Endpoint BFM.

The testbench uses a test driver module, **altpcietb_bfm_driver_avmm**, to exercise the target memory and DMA channel in the Endpoint BFM. The test driver module displays information from the Root Port Configuration Space registers, so that you can correlate to the parameters you specified using the parameter editor. The Endpoint model consists of an Endpoint variation combined with the DMA application described above.

Note: The Altera testbench and Root Port or Endpoint BFM provide a simple method to do basic testing of the Application Layer logic that interfaces to the variation. However, the testbench and Root Port BFM are not intended to be a substitute for a full verification environment. To thoroughly test your application, Altera suggests that you obtain commercially available PCI Express verification IP and tools, or do your own extensive hardware testing or both.

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Your Application Layer design may need to handle at least the following scenarios that are not possible to create with the Altera testbench and the Root Port BFM:

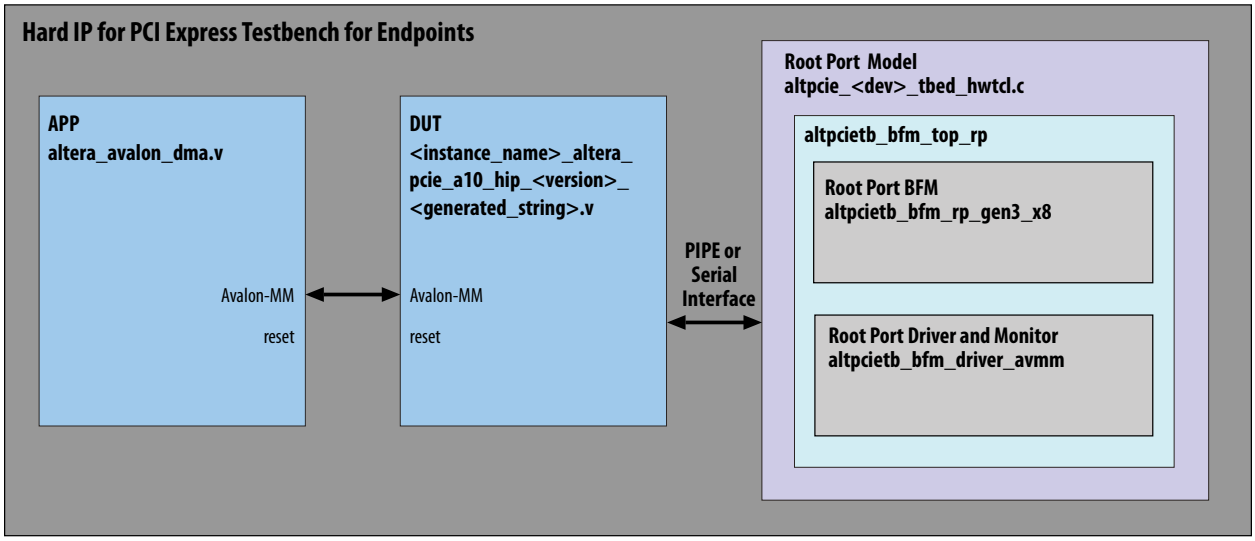
- It is unable to generate or receive Vendor Defined Messages. Some systems generate Vendor Defined Messages and the Application Layer must be designed to process them. The Hard IP block passes these messages on to the Application Layer which, in most cases should ignore them.
- It can only handle received read requests that are less than or equal to the currently set **Maximum payload size** option specified under **PCI Express/PCI Capabilities** heading under the **Device** tab using the parameter editor. Many systems are capable of handling larger read requests that are then returned in multiple completions.
- It always returns a single completion for every read request. Some systems split completions on every 64-byte address boundary.
- It always returns completions in the same order the read requests were issued. Some systems generate the completions out-of-order.
- It is unable to generate zero-length read requests that some systems generate as flush requests following some write transactions. The Application Layer must be capable of generating the completions to the zero length read requests.
- It uses fixed credit allocation.
- It does not support parity.
- It does not support multi-function designs which are available when using Configuration Space Bypass mode or Single Root I/O Virtualization (SR-IOV).

Arria 10 Avalon-MM Endpoint Testbench

After you install the Quartus II software, you can copy any of the example designs from the ***<install_dir>*/ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10** directory. You can generate the testbench from the example design as was shown in *Getting Started with the Arria 10 Hard IP for PCI Express*.

This testbench simulates up to an ×8 PCI Express link using either the PIPE interfaces of the Root Port and Endpoints or the serial PCI Express interface. The testbench design does not allow more than one PCI Express link to be simulated at a time. The following figure presents a high level view of the design example.

Figure 14-1: Design Example for Endpoint Designs



The top-level of the testbench instantiates the following main modules:

- <qsys_systemname>— This is the example Endpoint design. For more information about this module, refer to *DMA Design Examples*.
- **altpcie_a10_tbed_hwtcl.v**—This is the Root Port PCI Express BFM. For more information about this module, refer to *Root Port BFM*.
- **altera_avalon_dma**—This module drives transactions to the Root Port BFM. This is the module that you modify to vary the transactions sent to the example Endpoint design or your own design. For more information about this module, refer to *Root Port Design Example*.

In addition, the testbench has routines that perform the following tasks:

- Generates the reference clock for the Endpoint at the required frequency.
- Provides a PCI Express reset at start up.

Note: Before running the testbench, you should set the following parameters in **<*instantiation_name*>_tb/sim/<*instantiation_name*>_tb.v**:

- `serial_sim_hwtcl`: Set to 1 for serial simulation and 0 for PIPE simulation.
- `enable_pipe32_phyip_ser_driver_hwtcl`: Set to 0 for serial simulation and 1 for PIPE simulation.

Related Information

Getting Started with the Avalon-MM Arria 10 Hard IP for PCI Express on page 2-1

Arria 10 Avalon-MM Root Port Testbench

This testbench simulates up to an ×8 PCI Express link using either the PIPE interfaces or the serial PCI Express interface of the Root Port and Endpoints. The testbench design does not allow more than one PCI Express link to be simulated at a time. The top-level of the testbench instantiates four main modules:

- *<qsys_systemname>*— Name of Root Port This is the example Root Port design. For more information about this module, refer to *Root Port Design Example*.
- **altpcietb_bfm_ep_example_chaining_pipen1b**—This is the Endpoint PCI Express mode described in the section *DMA Design Examples*.
- **altpcietb_pipe_phy**—There are eight instances of this module, one per lane. These modules connect the PIPE MAC layer interfaces of the Root Port and the Endpoint. The module mimics the behavior of the PIPE PHY layer to both MAC interfaces.
- **altpcietb_bfm_driver_rp**—This module drives transactions to the Root Port BFM. This is the module that you modify to vary the transactions sent to the example Endpoint design or your own design. For more information about this module, see *Test Driver Module*.

The testbench has routines that perform the following tasks:

- Generates the reference clock for the Endpoint at the required frequency.
- Provides a reset at start up.

Note: Before running the testbench, you should set the following parameters:

- `serial_sim_hwtcl`: Set this parameter in **<instantiation name>_tb.v** . This parameter controls whether the testbench simulates in PIPE mode or serial mode. When is set to 0, the simulation runs in PIPE mode; when set to 1, it runs in serial mode. Although the `serial_sim_hwtcl` parameter is available in other files, if you set this parameter at the lower level, then it will get overwritten by the **tb.v** level.
- `serial_sim_hwtcl`: Set to 1 for serial simulation and 0 for PIPE simulation.
- `enable_pipe32_sim_hwtcl`: Set to 0 for serial simulation and 1 for PIPE simulation.

Endpoint Design Example

This design example comprises native Endpoint, a DMA application and a Root Port. The write DMA module implements write operations from the Endpoint memory to the root complex (RC) memory. The read DMA implements read operations from the RC memory to the Endpoint memory. The DMA and Endpoint support simultaneous read and write transactions.

When operating on a hardware platform, the DMA is typically controlled by a software application running on the root complex processor. In simulation, the generated testbench, along with this design example, provides a BFM driver module in Verilog HDL that controls the DMA operations. Because the example relies on no other hardware interface than the PCI Express link, you can use the design example for the initial hardware validation of your system.

The end point or Root Port variant is generated in the language (Verilog HDL or VHDL) that you selected for the variation file. The testbench files are only generated in Verilog HDL in the current release. If you choose to use VHDL for your variant, you must have a mixed-language simulator to run this testbench.

Note: The DMA design example requires setting BAR 2 or BAR 3 to a minimum of 256 bytes. To run the DMA tests using MSI, you must set the **Number of MSI messages requested** parameter under the **PCI Express/PCI Capabilities** page to at least 2.

The DMA design example uses an architecture capable of transferring a large amount of fragmented memory without accessing the DMA registers for every memory block. For each block of memory to be transferred, the DMA design example uses a descriptor table containing the following information:

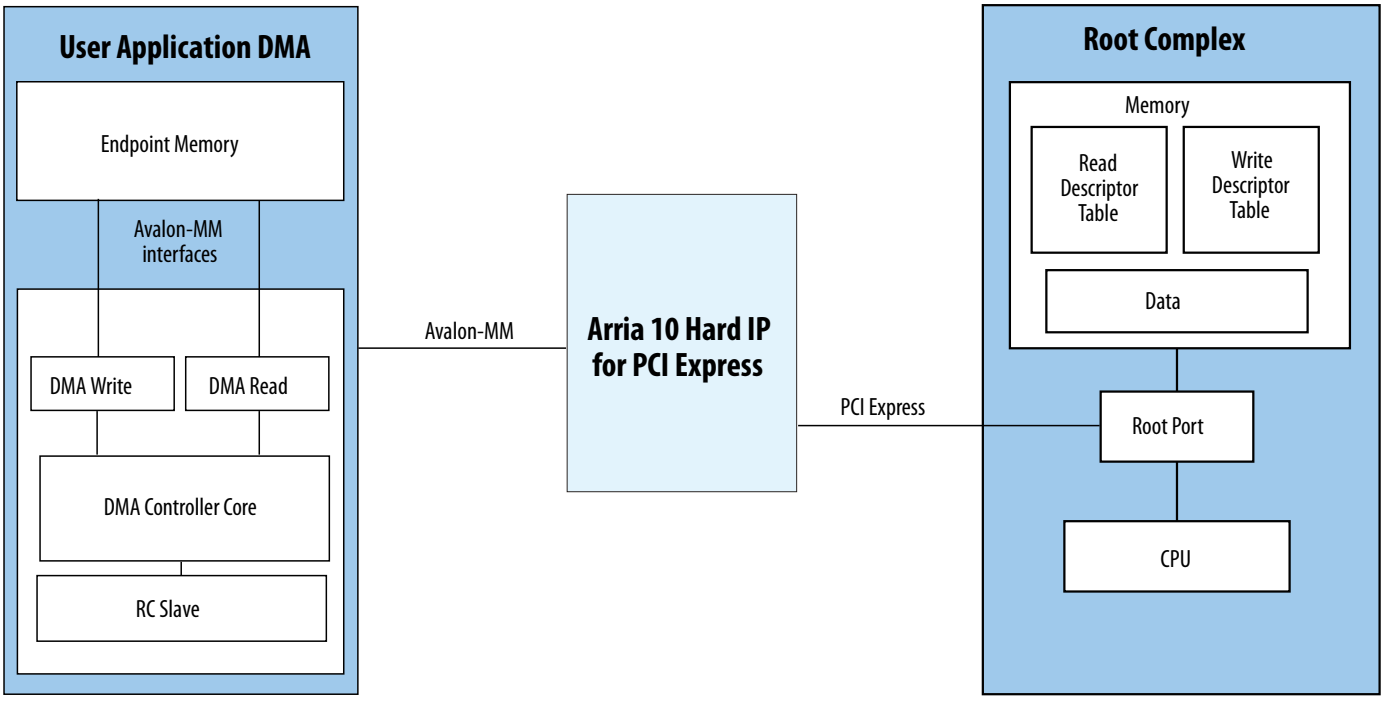
- Length of the transfer
- Address of the source
- Address of the destination
- Control bits to set the handshaking behavior between the software application or BFM driver and the DMA module

Note: The DMA design example only supports dword-aligned accesses. The DMA design example does not support ECRC forwarding.

The BFM driver writes the descriptor tables into BFM shared memory, from which the DMA design engine continuously collects the descriptor tables for DMA read, DMA write, or both. At the beginning of the transfer, the BFM programs the Endpoint DMA control register. The DMA control register indicates the total number of descriptor tables and the BFM shared memory address of the first descriptor table. After programming the DMA control register, the DMA engine continuously fetches descriptors from the BFM shared memory for both DMA reads and DMA writes, and then performs the data transfer for each descriptor.

The following figure shows a block diagram of the design example connected to an external RC CPU.

Figure 14-2: Top-Level DMA Example for Simulation



The block diagram contains the following elements:

- The DMA design example connects to the Avalon-MM interface of the Arria 10 Hard IP for PCI Express. The connections consist of the following interfaces:
  - The Avalon-MM RX master receives TLP header and data information from the Hard IP block
  - The Avalon-MM TX slave transmits TLP header and data information to the Hard IP block
  - The Avalon-MM control register access (CRA) IRQ port requests MSI interrupts from the Hard IP block
  - The sideband signal bus carries static information such as configuration information
- The descriptor tables of the DMA read and the DMA write are located in the BFM shared memory.
- A RC CPU and associated PCI Express PHY link to the Endpoint design example, using a Root Port.

The example Endpoint design Application Layer accomplishes the following objectives:

- Shows you how to interface to the Arria 10 Hard IP for PCI Express using the Avalon-MM protocol.
- Provides a DMA channel that initiates memory read and write transactions on the PCI Express link.

The following modules are included in the design example and located in the subdirectory **<*qsys_systemname*>_tb/<*qsys_system_name*>_tb/altera_pcie_<*a10*>_tbed_<*quartus_ver*>/sim** :

- <*qsys_systemname*> —This module is the top level of the example Endpoint design that you use for simulation.

  This module provides both PIPE and serial interfaces for the simulation environment. This module has a `test_in` debug ports. Refer to *Test Signals* which allow you to monitor and control internal states of the Hard IP.

  For synthesis, the top level module is **<*qsys_systemname*>/synth/**. This module instantiates the top-level module and propagates only a small sub-set of the test ports to the external I/Os. These test ports can be used in your design.
- <*variation name*>**.v** or <*variation name*>**.vhd**— Because Altera provides sample parameterizations, you may have to edit one of the provided examples to create a simulation that matches your requirements.

The DMA design example hierarchy consists of these components:

- A DMA read and a DMA write module
- An on-chip Endpoint memory (Avalon-MM slave) which uses two Avalon-MM interfaces for each engine

The RC slave module is used primarily for downstream transactions which target the Endpoint on-chip buffer memory. These target memory transactions bypass the DMA engines. In addition, the RC slave module monitors performance and acknowledges incoming message TLPs.

Related Information

Embedded Peripheral IP User Guide
For more information about the DMA Controller Qsys IP Core.

BAR/Address Map

The design example maps received memory transactions to either the target memory block or the control register block based on which BAR the transaction matches. There are multiple BARs that map to each of

these blocks to maximize interoperability with different variation files. The following table shows the mapping.

Table 14-1: BAR Map

Memory BAR	Mapping
32-bit BAR0 32-bit BAR1 64-bit BAR1:0	Maps to 32 KByte target memory block. Use the rc_slave module to bypass the chaining DMA.
32-bit BAR2 32-bit BAR3 64-bit BAR3:2	Maps to DMA Read and DMA write control and status registers, a minimum of 256 bytes.
32-bit BAR4 32-bit BAR5 64-bit BAR5:4	Maps to 32 KByte target memory block. Use the rc_slave module to bypass the chaining DMA.
Expansion ROM BAR	Not implemented by design example; behavior is unpredictable.
I/O Space BAR (any)	Not implemented by design example; behavior is unpredictable.

Avalon-MM Test Driver Module

The BFM driver module, **altpcietb_bfm_driver_avmm.v** is configured to test the DMA example Endpoint design. The BFM driver module configures the Endpoint Configuration Space registers and then tests the example Endpoint DMA channel. This file is stored in the **<*variation_name*>_tb/altera_pcie_<*dev*>_tbed_<*quartus_ver*>/sim/** directory.

The BFM test driver module performs the following steps in sequence:

1. Configures the Root Port and Endpoint Configuration Spaces, which the BFM test driver module does by calling the procedure `ebfm_cfg_rp_ep`, which is part of **altpcietb_bfm_configure**.
2. Finds a suitable BAR to access the example Endpoint design Control Register space. Either BARs 2 or 3 must be at least a 256-byte memory BAR to perform the DMA channel test. The `find_mem_bar` procedure in the **altpcietb_bfm_driver_avmm** does this.
3. If a suitable BAR is found in the previous step, the driver performs the following tasks:
   a. DMA read—The driver programs the DMA to read data from the BFM shared memory into the Endpoint memory. The descriptor control fields specify for the DMA to issue an MSI when the last descriptor has completed.
   a. DMA write—The driver programs the DMA to write the data from its Endpoint memory back to the BFM shared memory. The descriptor control fields are specified so that the DMA completes the following steps to indicate transfer completion:

- The DMA issues an MSI when the last descriptor has completed.
- The data written back to BFM is checked against the data that was read from the BFM.
- The driver programs the DMA to perform a test that demonstrates downstream access of the DMA Endpoint memory.

Note: Edit this file if you want to add your own custom PCIe transactions. Insert your own custom function after the `find_mem_bar` function. You can use the functions in the *BFM Procedures and Functions* section.

Related Information

BFM Procedures and Functions on page 14-22

DMA Write Cycles

The procedure `dma_wr_test` used for DMA writes uses the following steps:

1. Configures the BFM shared memory. Configuration is accomplished with three descriptor tables described below.

Table 14-2: Write Descriptor 0

	Offset in BFM in Shared Memory	Value	Description
DW0	0x810	82	Transfer length in dwords and control bits as described in *Bit Definitions for the Control Field in the DMA Write Control Register and DMA Read Control Register*.
DW1	0x814	3	Endpoint address
DW2	0x818	0	BFM shared memory data buffer 0 upper address value
DW3	0x81c	0x1800	BFM shared memory data buffer 1 lower address value
Data Buffer 0	0x1800	Increment by 1 from 0x1515_0001	Data content in the BFM shared memory from address: 0x01800–0x1840

Table 14-3: Write Descriptor 1

	Offset in BFM Shared Memory	Value	Description
DW0	0x820	1,024	Transfer length in dwords and control bits as described in *Bit Definitions for the Control Field in the DMA Write Control Register and DMA Read Control Register* .
DW1	0x824	0	Endpoint address

	Offset in BFM Shared Memory	Value	Description
DW2	0x828	0	BFM shared memory data buffer 1 upper address value
DW3	0x82c	0x2800	BFM shared memory data buffer 1 lower address value
Data Buffer 1	0x02800	Increment by 1 from 0x2525_0001	Data content in the BFM shared memory from address: 0x02800

Table 14-4: Write Descriptor 2

	Offset in BFM Shared Memory	Value	Description
DW0	0x830	644	Transfer length in dwords and control bits as described in *Bit Definitions for the Control Field in the DMA Write Control Register and DMA Read Control Register*.
DW1	0x834	0	Endpoint address
DW2	0x838	0	BFM shared memory data buffer 2 upper address value
DW3	0x83c	0x057A0	BFM shared memory data buffer 2 lower address value
Data Buffer 2	0x057A0	Increment by 1 from 0x3535_0001	Data content in the BFM shared memory from address: 0x057A0

2. Sets up the chaining DMA descriptor header and starts the transfer data from the Endpoint memory to the BFM shared memory. The transfer calls the procedure `dma_set_header` which writes four dwords, DW0:DW3, into the DMA write register module.

Table 14-5: DMA Control Register Setup for DMA Write

	Offset in DMA Control Register (BAR2)	Value	Description
DW0	0x0	3	Number of descriptors and control bits as described in *Chaining DMA Control Register Definitions*.
DW1	0x4	0	BFM shared memory descriptor table upper address value
DW2	0x8	0x800	BFM shared memory descriptor table lower address value

	Offset in DMA Control Register (BAR2)	Value	Description
DW3	0xc	2	Last valid descriptor

After writing the last dword, DW3, of the descriptor header, the DMA write starts the three subsequent data transfers.

3. Waits for the DMA write completion by polling the BFM share memory location 0x80c, where the DMA write engine is updating the value of the number of completed descriptor. Calls the procedures `rcmem_poll` and `msi_poll` to determine when the DMA write transfers have completed.

DMA Read Cycles

The procedure `dma_rd_test` used for DMA read uses the following three steps:

1. Configures the BFM shared memory with a call to the procedure `dma_set_rd_desc_data` which sets the following three descriptor tables. .

Table 14-6: Read Descriptor 0

	Offset in BFM Shared Memory	Value	Description
DW0	0x910	82	Transfer length in dwords and control bits as described in on page 18–15
DW1	0x914	3	Endpoint address value
DW2	0x918	0	BFM shared memory data buffer 0 upper address value
DW3	0x91c	0x8DF0	BFM shared memory data buffer 0 lower address value
Data Buffer 0	0x8DF0	Increment by 1 from 0xAAA0_0001	Data content in the BFM shared memory from address: 0x89F0

Table 14-7: Read Descriptor 1

	Offset in BFM Shared Memory	Value	Description
DW0	0x920	1,024	Transfer length in dwords and control bits as described in on page 18–15
DW1	0x924	0	Endpoint address value

	Offset in BFM Shared Memory	Value	Description
DW2	0x928	10	BFM shared memory data buffer 1 upper address value
DW3	0x92c	0x10900	BFM shared memory data buffer 1 lower address value
Data Buffer 1	0x10900	Increment by 1 from 0xBBBB_0001	Data content in the BFM shared memory from address: 0x10900

Table 14-8: Read Descriptor 2

	Offset in BFM Shared Memory	Value	Description
DW0	0x930	644	Transfer length in dwords and control bits as described in on page 18–15
DW1	0x934	0	Endpoint address value
DW2	0x938	0	BFM shared memory upper address value
DW3	0x93c	0x20EF0	BFM shared memory lower address value
Data Buffer 2	0x20EF0	Increment by 1 from 0xCCCC_0001	Data content in the BFM shared memory from address: 0x20EF0

2. Sets up the chaining DMA descriptor header and starts the transfer data from the BFM shared memory to the Endpoint memory by calling the procedure `dma_set_header` which writes four dwords, DW0:DW3 into the DMA read register module.

Table 14-9: DMA Control Register Setup for DMA Read

	Offset in DMA Control Registers (BAR2)	Value	Description
DW0	0x0	3	Number of descriptors and control bits as described in *Chaining DMA Control Register Definitions*.
DW1	0x14	0	BFM shared memory upper address value
DW2	0x18	0x900	BFM shared memory lower address value

	Offset in DMA Control Registers (BAR2)	Value	Description
DW3	0x1c	2	Last descriptor written

After writing the last dword of the Descriptor header (DW3), the DMA read starts the three subsequent data transfers.

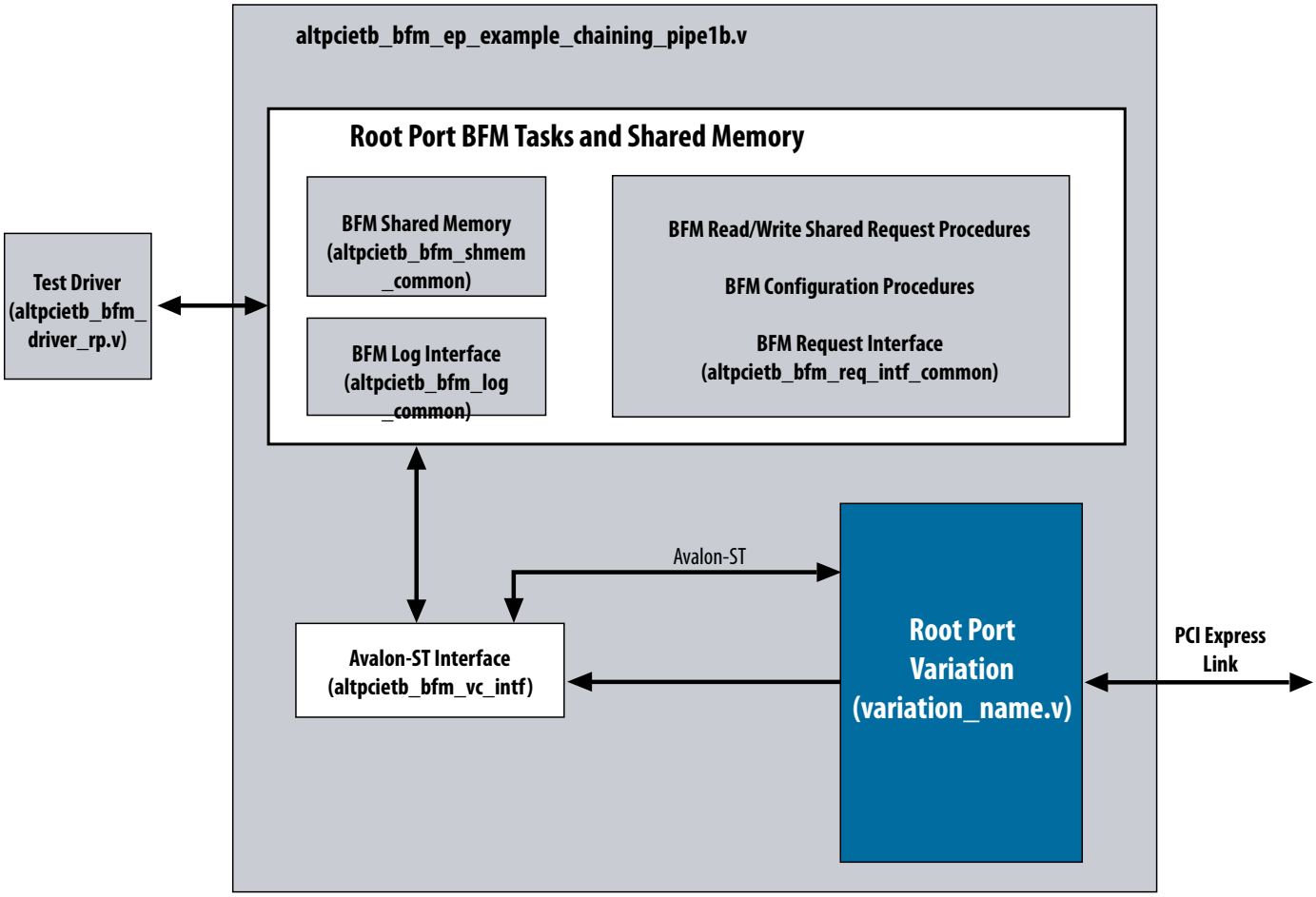
3. Waits for the DMA read completion by polling the BFM shared memory location 0x90c, where the DMA read engine is updating the value of the number of completed descriptors. Calls the procedures `rcmem_poll` and `msi_poll` to determine when the DMA read transfers have completed.

Avalon-MM Root Port Design Example

The design example includes the following primary components:

- Root Port variation (<*qsys_systemname*>.
- Avalon-ST Interfaces (**altpcietb_bfm_vc_intf_ast**)—handles the transfer of TLP requests and completions to and from the Arria 10 Hard IP for PCI Express variation using the Avalon-ST interface.
- Root Port BFM tasks—contains the high-level tasks called by the test driver, low-level tasks that request PCI Express transfers from **altpcietb_bfm_vc_intf_ast**, the Root Port memory space, and simulation functions such as displaying messages and stopping simulation.
- Test Driver (**altpcietb_bfm_driver_rp.v**)—the DMA Endpoint test driver which configures the Root Port and Endpoint for DMA transfer and checks for the successful transfer of data. Refer to the *Test Driver Module*for a detailed description.

Figure 14-3: Root Port Design Example



You can use the example Root Port design for Verilog HDL simulation. All of the modules necessary to implement the example design with the variation file are contained in **altpcietb_bfm_ep_example_chaining_pipen1b.v**.

The top-level of the testbench instantiates the following key files:

- **altlpcietb_bfm_top_ep.v**— this is the Endpoint BFM. This file also instantiates the SERDES and PIPE interface.
- **altpcietb_pipe_phy.v**—used to simulate the PIPE interface.
- **altp cietb_bfm_ep_example_chaining_pipen1b.v**—the top-level of the Root Port design example that you use for simulation. This module instantiates the Root Port variation, <*variation_name*> **.v**, and the Root Port application **altpcietb_bfm_vc_intf** _<*application_width*> . This module provides both PIPE and serial interfaces for the simulation environment. This module has two debug ports named `test_out_icm_`(which is the `test_out` signal from the Hard IP) and `test_in` which allows you to monitor and control internal states of the Hard IP variation.
- **altpcietb_bfm_vc_intf_ast.v**—a wrapper module which instantiates either **altpcietb_vc_intf_64** or **altpcietb_vc_intf_** <*application_width*> based on the type of Avalon-ST interface that is generated.
- **altpcietb_vc_intf_** _<*application_width*> **.v**—provide the interface between the Arria 10 Hard IP for PCI Express variant and the Root Port BFM tasks. They provide the same function as the **altpcietb_bfm_vc_intf.v** module, transmitting requests and handling completions. Refer to the *Root Port BFM* for a full description of this function. This version uses Avalon-ST signalling with either a 64- or 128-bit data bus interface.
- **altpcierd_tl_cfg_sample.v**—accesses Configuration Space signals from the variant. Refer to the *DMA Design Examples* for a description of this module.

Files in subdirectory **<*variant_name*>_tb/altera_pcie_<*dev*>_tbed_<*quartus_ver*>/sim/**:

- **altpcietb_bfm_ep_example_chaining_pipen1b.v**—the simulation model for the DMA Endpoint.
- **altpcietb_bfm_driver_rp.v**–this file contains the functions to implement the shared memory space, PCI Express reads and writes, initialize the Configuration Space registers, log and display simulation messages, and define global constants.

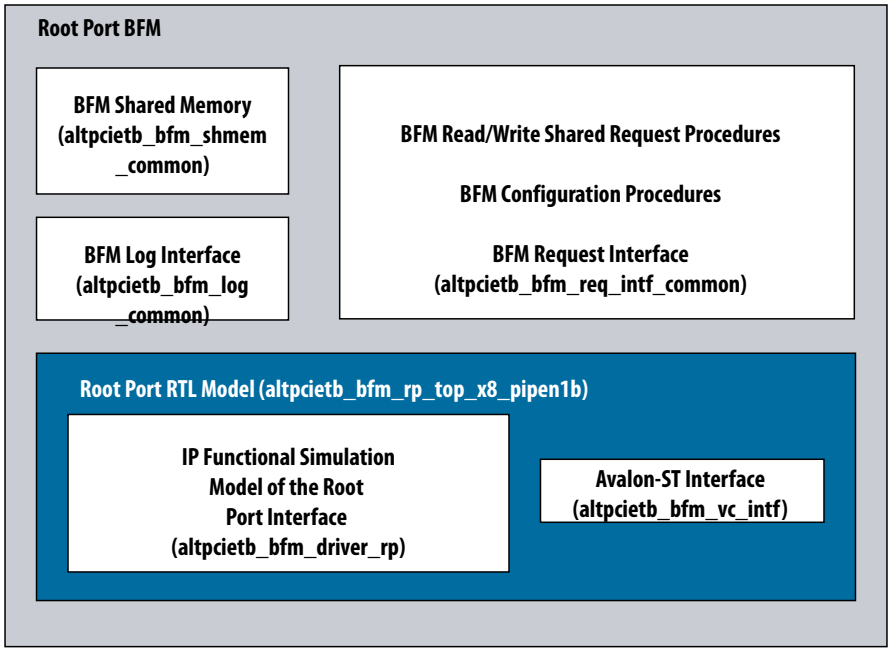
Related Information

Avalon-MM Test Driver Module on page 14-7

Root Port BFM

The basic Root Port BFM provides Verilog HDL task-based interface for requesting transactions that are issued to the PCI Express link. The Root Port BFM also handles requests received from the PCI Express link. The following figure provides an overview of the Root Port BFM.

Figure 14-4: Root Port BFM


Root Port BFM
BFM Shared Memory (altpcietb_bfm_shmem_common)
BFM Log Interface (altpcietb_bfm_log_common)
BFM Read/Write Shared Request Procedures
BFM Configuration Procedures
BFM Request Interface (altpcietb_bfm_req_intf_common)
Root Port RTL Model (altpcietb_bfm_rp_top_x8_pipen1b)
IP Functional Simulation Model of the Root Port Interface (altpcietb_bfm_driver_rp)
Avalon-ST Interface (altpcietb_bfm_vc_intf)

The functionality of each of the modules included is explained below.

- BFM shared memory (**altpcietb_bfm_shmem_common** Verilog HDL include file)—The Root Port BFM is based on the BFM memory that is used for the following purposes:
- Storing data received with all completions from the PCI Express link.
- Storing data received with all write transactions received from the PCI Express link.
- Sourcing data for all completions in response to read transactions received from the PCI Express link.
- Sourcing data for most write transactions issued to the PCI Express link. The only exception is certain BFM write procedures that have a four-byte field of write data passed in the call.
- Storing a data structure that contains the sizes of and the values programmed in the BARs of the Endpoint.

A set of procedures is provided to read, write, fill, and check the shared memory from the BFM driver. For details on these procedures, see *BFM Shared Memory Access Procedures*.

- BFM Read/Write Request Functions(**altpcietb_bfm_driver_rp.v**)—These functions provide the basic BFM calls for PCI Express read and write requests. For details on these procedures, refer to *BFM Read and Write Procedures*.
- BFM Configuration Functions(**altpcietb_bfm_driver_rp.v**)—These functions provide the BFM calls to request configuration of the PCI Express link and the Endpoint Configuration Space registers. For details on these procedures and functions, refer to *BFM Configuration Procedures*.
- BFM Log Interface(**altpcietb_bfm_driver_rp.v**)—The BFM log functions provides routines for writing commonly formatted messages to the simulator standard output and optionally to a log file. It also provides controls that stop simulation on errors. For details on these procedures, refer to *BFM Log and Message Procedures*.
- BFM Request Interface(**altpcietb_bfm_driver_rp.v**)—This interface provides the low-level interface between the `altpcietb_bfm_rdwr` and `altpcietb_bfm_configure` procedures or functions and the Root Port RTL Model. This interface stores a write-protected data structure containing the sizes and the values programmed in the BAR registers of the Endpoint, as well as, other critical data used for internal BFM management. You do not need to access these files directly to adapt the testbench to test your Endpoint application.
- Avalon-ST Interfaces (**altpcietb_bfm_vc_intf.v**)—These interface modules handle the Root Port interface model. They take requests from the BFM request interface and generate the required PCI Express transactions. They handle completions received from the PCI Express link and notify the BFM request interface when requests are complete. Additionally, they handle any requests received from the PCI Express link, and store or fetch data from the shared memory before generating the required completions.

Related Information

- **Test Signals** on page 5-19
- **BFM Shared Memory Access Procedures** on page 14-31

BFM Memory Map

The BFM shared memory is configured to be two MBytes. The BFM shared memory is mapped into the first two MBytes of I/O space and also the first two MBytes of memory space. When the Endpoint application generates an I/O or memory transaction in this range, the BFM reads or writes the shared memory.

Configuration Space Bus and Device Numbering

The Root Port interface is assigned to be device number 0 on internal bus number 0. The Endpoint can be assigned to be any device number on any bus number (greater than 0) through the call to procedure `ebfm_cfg_rp_ep`. The specified bus number is assigned to be the secondary bus in the Root Port Configuration Space.

Configuration of Root Port and Endpoint

Before you issue transactions to the Endpoint, you must configure the Root Port and Endpoint Configuration Space registers. To configure these registers, call the procedure `ebfm_cfg_rp_ep`, which is included in **altpcietb_bfm_configure.v**.

The ebfm_cfg_rp_ep executes the following steps to initialize the Configuration Space:

1. Sets the Root Port Configuration Space to enable the Root Port to send transactions on the PCI Express link.
2. Sets the Root Port and Endpoint PCI Express Capability Device Control registers as follows:
   a. Disables Error Reporting in both the Root Port and Endpoint. BFM does not have error handling capability.
   b. Enables Relaxed Ordering in both Root Port and Endpoint.
   c. Enables Extended Tags for the Endpoint, if the Endpoint has that capability.
   d. Disables Phantom Functions, Aux Power PM, and No Snoop in both the Root Port and Endpoint.
   e. Sets the Max Payload Size to what the Endpoint supports because the Root Port supports the maximum payload size.
   f. Sets the Root Port Max Read Request Size to 4 KBytes because the example Endpoint design supports breaking the read into as many completions as necessary.
   g. Sets the Endpoint Max Read Request Size equal to the Max Payload Size because the Root Port does not support breaking the read request into multiple completions.
3. Assigns values to all the Endpoint BAR registers. The BAR addresses are assigned by the algorithm outlined below.
   a. I/O BARs are assigned smallest to largest starting just above the ending address of BFM shared memory in I/O space and continuing as needed throughout a full 32-bit I/O space.
   b. The 32-bit non-prefetchable memory BARs are assigned smallest to largest, starting just above the ending address of BFM shared memory in memory space and continuing as needed throughout a full 32-bit memory space.
   c. Assignment of the 32-bit prefetchable and 64-bit prefetchable memory BARS are based on the value of the addr_map_4GB_limit input to the ebfm_cfg_rp_ep. The default value of the addr_map_4GB_limit is 0.

      If the addr_map_4GB_limit input to the ebfm_cfg_rp_ep is set to 0, then the 32-bit prefetchable memory BARs are assigned largest to smallest, starting at the top of 32-bit memory space and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.

      However, if the addr_map_4GB_limit input is set to 1, the address map is limited to 4 GByte, the 32-bit and 64-bit prefetchable memory BARs are assigned largest to smallest, starting at the top of the 32-bit memory space and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.
   d. If the addr_map_4GB_limit input to the ebfm_cfg_rp_ep is set to 0, then the 64-bit prefetchable memory BARs are assigned smallest to largest starting at the 4 GByte address assigning memory ascending above the 4 GByte limit throughout the full 64-bit memory space.

      If the addr_map_4 GB_limit input to the ebfm_cfg_rp_ep is set to 1, then the 32-bit and the 64-bit prefetchable memory BARs are assigned largest to smallest starting at the 4 GByte address and assigning memory by descending below the 4 GByte address to addresses memory as needed down to the ending address of the last 32-bit non-prefetchable BAR.

      The above algorithm cannot always assign values to all BARs when there are a few very large (1 GByte or greater) 32-bit BARs. Although assigning addresses to all BARs may be possible, a more complex algorithm would be required to effectively assign these addresses. However, such a

configuration is unlikely to be useful in real systems. If the procedure is unable to assign the BARs, it displays an error message and stops the simulation.

4. Based on the above BAR assignments, the Root Port Configuration Space address windows are assigned to encompass the valid BAR address ranges.
5. The Endpoint PCI control register is set to enable master transactions, memory address decoding, and I/O address decoding.

The `ebfm_cfg_rp_ep` procedure also sets up a `bar_table` data structure in BFM shared memory that lists the sizes and assigned addresses of all Endpoint BARs. This area of BFM shared memory is write-protected, which means any user write accesses to this area cause a fatal simulation error. This data structure is then used by subsequent BFM procedure calls to generate the full PCI Express addresses for read and write requests to particular offsets from a BAR. This procedure allows the testbench code that accesses the Endpoint Application Layer to be written to use offsets from a BAR and not have to keep track of the specific addresses assigned to the BAR. The following table shows how those offsets are used.

Table 14-10: BAR Table Structure

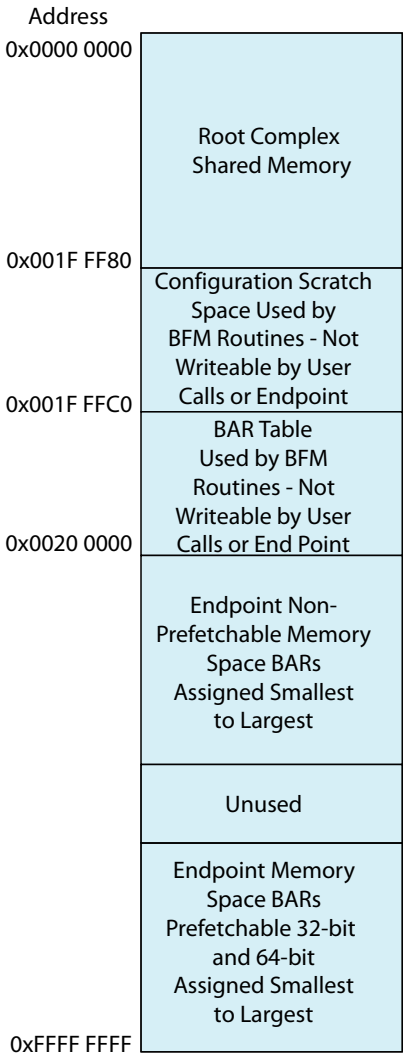
Offset (Bytes)	Description
+0	PCI Express address in BAR0
+4	PCI Express address in BAR1
+8	PCI Express address in BAR2
+12	PCI Express address in BAR3
+16	PCI Express address in BAR4
+20	PCI Express address in BAR5
+24	PCI Express address in Expansion ROM BAR
+28	Reserved
+32	BAR0 read back value after being written with all 1's (used to compute size)
+36	BAR1 read back value after being written with all 1's
+40	BAR2 read back value after being written with all 1's
+44	BAR3 read back value after being written with all 1's
+48	BAR4 read back value after being written with all 1's
+52	BAR5 read back value after being written with all 1's
+56	Expansion ROM BAR read back value after being written with all 1's

Offset (Bytes)	Description
+60	Reserved

The configuration routine does not configure any advanced PCI Express capabilities such as the AER capability.

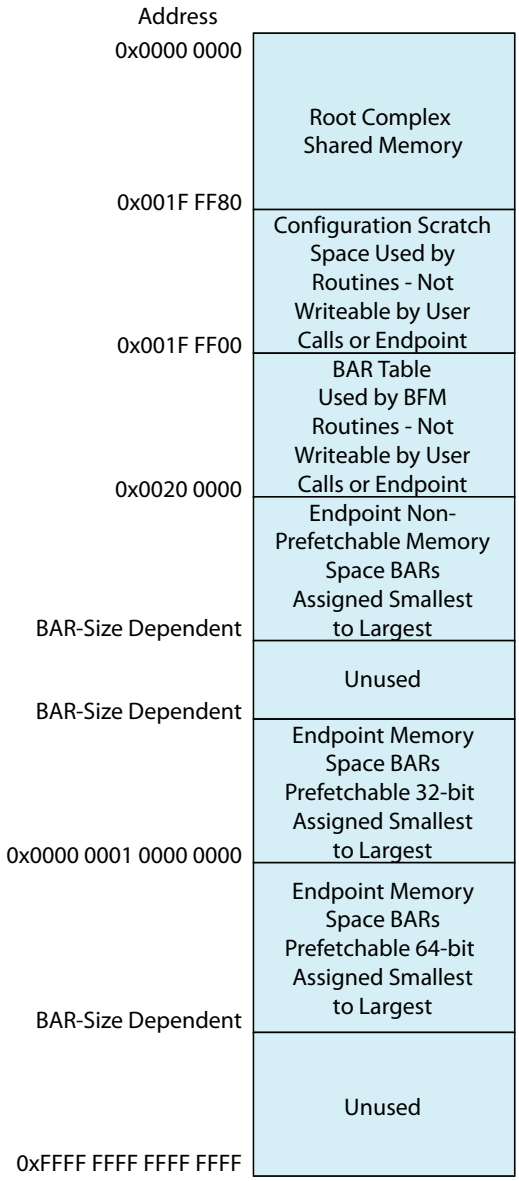
Besides the `ebfm_cfg_rp_ep` procedure in **altpcietb_bfm_driver_rp.v**, routines to read and write Endpoint Configuration Space registers directly are available in the Verilog HDL include file. After the `ebfm_cfg_rp_ep` procedure is run the PCI Express I/O and Memory Spaces have the layout as described in the following three figures. The memory space layout is dependent on the value of the **addr_map_4GB_limit** input parameter. If **addr_map_4GB_limit** is 1 the resulting memory space map is shown in the following figure.

Figure 14-5: Memory Space Layout—4 GByte Limit


Address
0x0000 0000
Root Complex Shared Memory
0x001F FF80
Configuration Scratch Space Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x001F FFC0
BAR Table Used by BFM Routines - Not Writeable by User Calls or End Point
0x0020 0000
Endpoint Non-Prefetchable Memory Space BARs Assigned Smallest to Largest
Unused
Endpoint Memory Space BARs Prefetchable 32-bit and 64-bit Assigned Smallest to Largest
0xFFFF FFFF

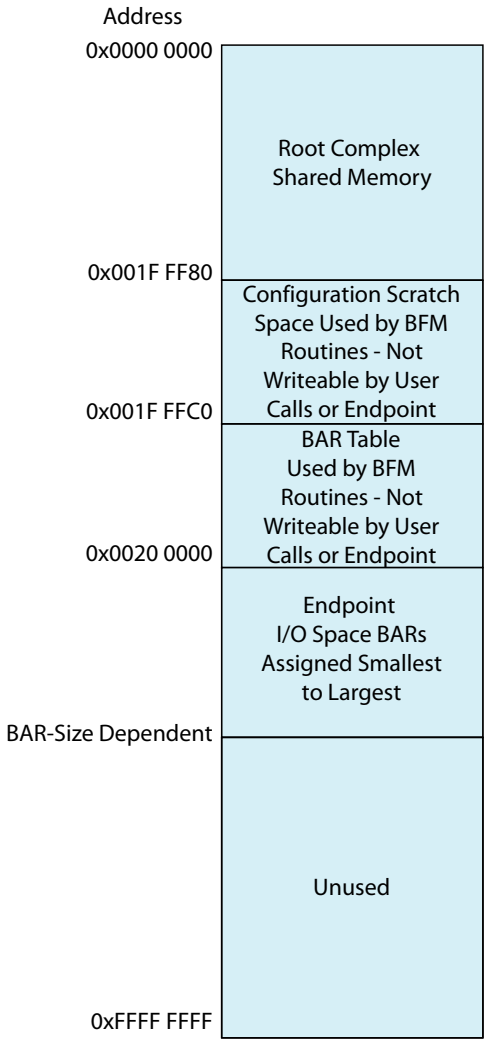
If **addr_map_4GB_limit** is 0, the resulting memory space map is shown in the following figure.

Figure 14-6: Memory Space Layout—No Limit


Address
0x0000 0000
Root Complex Shared Memory
0x001F FF80
Configuration Scratch Space Used by Routines - Not Writeable by User Calls or Endpoint
0x001F FF00
BAR Table Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x0020 0000
Endpoint Non-Prefetchable Memory Space BARs Assigned Smallest to Largest
BAR-Size Dependent
Unused
BAR-Size Dependent
Endpoint Memory Space BARs Prefetchable 32-bit Assigned Smallest to Largest
0x0000 0001 0000 0000
Endpoint Memory Space BARs Prefetchable 64-bit Assigned Smallest to Largest
BAR-Size Dependent
Unused
0xFFFF FFFF FFFF FFFF

The following figure shows the I/O address space.

Figure 14-7: I/O Address Space


Address
0x0000 0000
Root Complex Shared Memory
0x001F FF80
Configuration Scratch Space Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x001F FFC0
BAR Table Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x0020 0000
Endpoint I/O Space BARs Assigned Smallest to Largest
BAR-Size Dependent
Unused
0xFFFF FFFF

Issuing Read and Write Transactions to the Application Layer

Read and write transactions are issued to the Endpoint Application Layer by calling one of the `ebfm_bar` procedures in **altpcietb_bfm_driver_rp.v**. The procedures and functions listed below are available in the

Verilog HDL include file **altpcietb_bfm_driver_rp.v**. The complete list of available procedures and functions is as follows:

- `ebfm_barwr`—writes data from BFM shared memory to an offset from a specific Endpoint BAR. This procedure returns as soon as the request has been passed to the VC interface module for transmission.
- `ebfm_barwr_imm`—writes a maximum of four bytes of immediate data (passed in a procedure call) to an offset from a specific Endpoint BAR. This procedure returns as soon as the request has been passed to the VC interface module for transmission.
- `ebfm_barrd_wait`—reads data from an offset of a specific Endpoint BAR and stores it in BFM shared memory. This procedure blocks waiting for the completion data to be returned before returning control to the caller.
- `ebfm_barrd_nowt`—reads data from an offset of a specific Endpoint BAR and stores it in the BFM shared memory. This procedure returns as soon as the request has been passed to the VC interface module for transmission, allowing subsequent reads to be issued in the interim.

These routines take as parameters a BAR number to access the memory space and the BFM shared memory address of the `bar_table` data structure that was set up by the `ebfm_cfg_rp_ep` procedure. (Refer to *Configuration of Root Port and Endpoint*.) Using these parameters simplifies the BFM test driver routines that access an offset from a specific BAR and eliminates calculating the addresses assigned to the specified BAR.

The Root Port BFM does not support accesses to Endpoint I/O space BARs.

Related Information

Configuration of Root Port and Endpoint on page 14-16

BFM Procedures and Functions

The BFM includes procedures, functions, and tasks to drive Endpoint application testing. It also includes procedures to run the chaining DMA design example.

The BFM read and write procedures read and write data among BFM shared memory, Endpoint BARs, and specified configuration registers. The procedures and functions are available in the Verilog HDL. They are in the include file **altpcietb_bfm_driver_avmm.v**. These procedures and functions support issuing memory and configuration transactions on the PCI Express link.

ebfm_barwr Procedure

The `ebfm_barwr` procedure writes a block of data from BFM shared memory to an offset from the specified Endpoint BAR. The length can be longer than the configured `MAXIMUM_PAYLOAD_SIZE`; the procedure breaks the request up into multiple transactions as needed. This routine returns as soon as the last transaction has been accepted by the VC interface module.

Location	altpcietb_bfm_rdwr.v
Syntax	`ebfm_barwr(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)`

Location	altpcietb_bfm_rdwr.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	lcladdr	BFM shared memory address of the data to be written.
	byte_len	Length, in bytes, of the data written. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	tclass	Traffic class used for the PCI Express transaction.

ebfm_barwr_imm Procedure

The ebfm_barwr_imm procedure writes up to four bytes of data to an offset from the specified Endpoint BAR.

Location	altpcietb_bfm_driver_rp.v
Syntax	ebfm_barwr_imm(bar_table, bar_num, pcie_offset, imm_data, byte_len, tclass)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	imm_data	Data to be written. In Verilog HDL, this argument is reg [31:0].In both languages, the bits written depend on the length as follows: Length Bits Written • 4: 31 downto 0 • 3: 23 downto 0 • 2: 15 downto 0 • 1: 7 downto 0
	byte_len	Length of the data to be written in bytes. Maximum length is 4 bytes.
	tclass	Traffic class to be used for the PCI Express transaction.

ebfm_barrd_wait Procedure

The ebfm_barrd_wait procedure reads a block of data from the offset of the specified Endpoint BAR and stores it in BFM shared memory. The length can be longer than the configured maximum read request size; the procedure breaks the request up into multiple transactions as needed. This procedure waits until all of the completion data is returned and places it in shared memory.

Location	**altpcietb_bfm_driver_rp.v**
Syntax	ebfm_barrd_wait(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)

Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	lcladdr	BFM shared memory address where the read data is stored.
	byte_len	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	tclass	Traffic class used for the PCI Express transaction.

ebfm_barrd_nowt Procedure

The ebfm_barrd_nowt procedure reads a block of data from the offset of the specified Endpoint BAR and stores the data in BFM shared memory. The length can be longer than the configured maximum read request size; the procedure breaks the request up into multiple transactions as needed. This routine returns as soon as the last read transaction has been accepted by the VC interface module, allowing subsequent reads to be issued immediately.

Location	altpcietb_b fm_driver_rp.v	
Syntax	ebfm_barrd_nowt(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	lcladdr	BFM shared memory address where the read data is stored.
	byte_len	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	tclass	Traffic Class to be used for the PCI Express transaction.

ebfm_cfgwr_imm_wait Procedure

The `ebfm_cfgwr_imm_wait` procedure writes up to four bytes of data to the specified configuration register. This procedure waits until the write completion has been returned.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_cfgwr_imm_wait(bus_num, dev_num, fnc_num, imm_regb_ad, regb_ln, imm_data, compl_status`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes. The `regb_ln` and the `regb_ad` arguments cannot cross a DWORD boundary.
	`imm_data`	Data to be written. This argument is `reg [31:0]`. The bits written depend on the length: • 4: 31 downto 0 • 3: 23 downto 0 • 2: 15 downto 0 • 1: 7 downto 0
	`compl_status`	This argument is `reg [2:0]`. This argument is the completion status as specified in the PCI Express specification. The following encodings are defined: • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

ebfm_cfgwr_imm_nowt Procedure

The `ebfm_cfgwr_imm_nowt` procedure writes up to four bytes of data to the specified configuration register. This procedure returns as soon as the VC interface module accepts the transaction, allowing other writes to be issued in the interim. Use this procedure only when successful completion status is expected.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_cfgwr_imm_nowt(bus_num, dev_num, fnc_num, imm_regb_adr, regb_len, imm_data)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes, The `regb_ln` the `regb_ad` arguments cannot cross a DWORD boundary.
	`imm_data`	Data to be written This argument is `reg [31:0]`. In both languages, the bits written depend on the length. The following encodes are defined. • 4: [31:0] • 3: [23:0] • 2: [15:0] • 1: [7:0]

ebfm_cfgrd_wait Procedure

The `ebfm_cfgrd_wait` procedure reads up to four bytes of data from the specified configuration register and stores the data in BFM shared memory. This procedure waits until the read completion has been returned.

Location	altpcietb_bfm_driver_rp.v
Syntax	`ebfm_cfgrd_wait(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr, compl_status)`

Location	altpcietb_bfm_driver_rp.v	
Arguments	bus_num	PCI Express bus number of the target device.
	dev_num	PCI Express device number of the target device.
	fnc_num	Function number in the target device to be accessed.
	regb_ad	Byte-specific address of the register to be written.
	regb_ln	Length, in bytes, of the data read. Maximum length is four bytes. The regb_ln and the regb_ad arguments cannot cross a DWORD boundary.
	lcladdr	BFM shared memory address of where the read data should be placed.
	compl_status	Completion status for the configuration transaction. This argument is reg [2:0]. In both languages, this is the completion status as specified in the PCI Express specification. The following encodings are defined. • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

ebfm_cfgrd_nowt Procedure

The ebfm_cfgrd_nowt procedure reads up to four bytes of data from the specified configuration register and stores the data in the BFM shared memory. This procedure returns as soon as the VC interface module has accepted the transaction, allowing other reads to be issued in the interim. Use this procedure only when successful completion status is expected and a subsequent read or write with a wait can be used to guarantee the completion of this operation.

Location	altpcietb_bfm_driver_rp.v
Syntax	ebfm_cfgrd_nowt(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bus_num	PCI Express bus number of the target device.
	dev_num	PCI Express device number of the target device.
	fnc_num	Function number in the target device to be accessed.
	regb_ad	Byte-specific address of the register to be written.
	regb_ln	Length, in bytes, of the data written. Maximum length is four bytes. The regb_ln and regb_ad arguments cannot cross a DWORD boundary.
	lcladdr	BFM shared memory address where the read data should be placed.

BFM Configuration Procedures

The BFM configuration procedures are available in **altpcietb_bfm_driver_rp.v**. These procedures support configuration of the Root Port and Endpoint Configuration Space registers.

All Verilog HDL arguments are type `integer` and are input-only unless specified otherwise.

ebfm_cfg_rp_ep Procedure

The `ebfm_cfg_rp_ep` procedure configures the Root Port and Endpoint Configuration Space registers for operation.

Location	altpcietb_bfm_driver_rp.v
Syntax	`ebfm_cfg_rp_ep(bar_table, ep_bus_num, ep_dev_num, rp_max_rd_req_size, display_ep_config, addr_map_4GB_limit)`

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. This routine populates the bar_table structure. The bar_table structure stores the size of each BAR and the address values assigned to each BAR. The address of the bar_table structure is passed to all subsequent read and write procedure calls that access an offset from a particular BAR.
	ep_bus_num	PCI Express bus number of the target device. This number can be any value greater than 0. The Root Port uses this as its secondary bus number.
	ep_dev_num	PCI Express device number of the target device. This number can be any value. The Endpoint is automatically assigned this value when it receives its first configuration transaction.
	rp_max_rd_req_size	Maximum read request size in bytes for reads issued by the Root Port. This parameter must be set to the maximum value supported by the Endpoint Application Layer. If the Application Layer only supports reads of the MAXIMUM_PAYLOAD_SIZE, then this can be set to 0 and the read request size will be set to the maximum payload size. Valid values for this argument are 0, 128, 256, 512, 1,024, 2,048 and 4,096.
	display_ep_config	When set to 1 many of the Endpoint Configuration Space registers are displayed after they have been initialized, causing some additional reads of registers that are not normally accessed during the configuration process such as the Device ID and Vendor ID.
	addr_map_4GB_limit	When set to 1 the address map of the simulation system will be limited to 4 GBytes. Any 64-bit BARs will be assigned below the 4 GByte limit.

ebfm_cfg_decode_bar Procedure

The ebfm_cfg_decode_bar procedure analyzes the information in the BAR table for the specified BAR and returns details about the BAR attributes.

Location	altpcietb_bfm_driver_rp.v
Syntax	ebfm_cfg_decode_bar(bar_table, bar_num, log2_size, is_mem, is_pref, is_64b)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory.
	bar_num	BAR number to analyze.
	log2_size	This argument is set by the procedure to the log base 2 of the size of the BAR. If the BAR is not enabled, this argument will be set to 0.
	is_mem	The procedure sets this argument to indicate if the BAR is a memory space BAR (1) or I/O Space BAR (0).
	is_pref	The procedure sets this argument to indicate if the BAR is a prefetchable BAR (1) or non-prefetchable BAR (0).
	is_64b	The procedure sets this argument to indicate if the BAR is a 64-bit BAR (1) or 32-bit BAR (0). This is set to 1 only for the lower numbered BAR of the pair.

BFM Shared Memory Access Procedures

The BFM shared memory access procedures and functions are in the Verilog HDL include file **altpcietb_bfm_driver.v**. These procedures and functions support accessing the BFM shared memory.

Shared Memory Constants

The following constants are defined in **altpcietb_bfm_driver.v**. They select a data pattern in the `shmem_fill` and `shmem_chk_ok` routines. These shared memory constants are all Verilog HDL type `integer`.

Table 14-11: Constants: Verilog HDL Type INTEGER

Constant	Description
SHMEM_FILL_ZEROS	Specifies a data pattern of all zeros
SHMEM_FILL_BYTE_INC	Specifies a data pattern of incrementing 8-bit bytes (0x00, 0x01, 0x02, etc.)
SHMEM_FILL_WORD_INC	Specifies a data pattern of incrementing 16-bit words (0x0000, 0x0001, 0x0002, etc.)
SHMEM_FILL_DWORD_INC	Specifies a data pattern of incrementing 32-bit dwords (0x00000000, 0x00000001, 0x00000002, etc.)

Constant	Description
SHMEM_FILL_QWORD_INC	Specifies a data pattern of incrementing 64-bit qwords (0x0000000000000000, 0x0000000000000001, 0x0000000000000002, etc.)
SHMEM_FILL_ONE	Specifies a data pattern of all ones

shmem_write

The shmem_write procedure writes data to the BFM shared memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	shmem_write(addr, data, leng)	
Arguments	addr	BFM shared memory starting address for writing data
	data	Data to write to BFM shared memory. This parameter is implemented as a 64-bit vector. leng is 1–8 bytes. Bits 7 downto 0 are written to the location specified by addr; bits 15 downto 8 are written to the addr+1 location, etc.
	length	Length, in bytes, of data written

shmem_read Function

The shmem_read function reads data to the BFM shared memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	data:= shmem_read(addr, leng)	
Arguments	addr	BFM shared memory starting address for reading data
	leng	Length, in bytes, of data read
Return	data	Data read from BFM shared memory. This parameter is implemented as a 64-bit vector. leng is 1- 8 bytes. If leng is less than 8 bytes, only the corresponding least significant bits of the returned data are valid. Bits 7 downto 0 are read from the location specified by addr; bits 15 downto 8 are read from the addr+1 location, etc.

shmem_display Verilog HDL Function

The `shmem_display` Verilog HDL function displays a block of data from the BFM shared memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	Verilog HDL: `dummy_return:=shmem_display(addr, leng, word_size, flag_addr, msg_type);`	
Arguments	`addr`	BFM shared memory starting address for displaying data.
	`leng`	Length, in bytes, of data to display.
	`word_size`	Size of the words to display. Groups individual bytes into words. Valid values are 1, 2, 4, and 8.
	`flag_addr`	Adds a <== flag to the end of the display line containing this address. Useful for marking specific data. Set to a value greater than 2**21 (size of BFM shared memory) to suppress the flag.
	`msg_type`	Specifies the message type to be displayed at the beginning of each line. See "BFM Log and Message Procedures" on page 18–37 for more information about message types. Set to one of the constants defined in Table 18–36 on page 18–41.

shmem_fill Procedure

The `shmem_fill` procedure fills a block of BFM shared memory with a specified data pattern.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`shmem_fill(addr, mode, leng, init)`	
Arguments	`addr`	BFM shared memory starting address for filling data.
	`mode`	Data pattern used for filling the data. Should be one of the constants defined in section *Shared Memory Constants*.
	`leng`	Length, in bytes, of data to fill. If the length is not a multiple of the incrementing data pattern width, then the last data pattern is truncated to fit.
	`init`	Initial data value used for incrementing data pattern modes. This argument is `reg [63:0]`. The necessary least significant bits are used for the data patterns that are smaller than 64 bits.

Related Information

Shared Memory Constants on page 14-31

shmem_chk_ok Function

The `shmem_chk_ok` function checks a block of BFM shared memory against a specified data pattern.

Location	altpcietb_bfm_shmem.v	
Syntax	`result:= shmem_chk_ok(addr, mode, leng, init, display_error)`	
Arguments	`addr`	BFM shared memory starting address for checking data.
	`mode`	Data pattern used for checking the data. Should be one of the constants defined in section "Shared Memory Constants" on page 18–35.
	`leng`	Length, in bytes, of data to check.
	`init`	This argument is `reg [63:0]`.The necessary least significant bits are used for the data patterns that are smaller than 64-bits.
	`display_error`	When set to 1, this argument displays the mis-comparing data on the simulator standard output.
Return	`Result`	Result is 1-bit. • 1'b1 — Data patterns compared successfully • 1'b0 — Data patterns did not compare successfully

BFM Log and Message Procedures

The following procedures and functions are available in the Verilog HDL include file **altpcietb_bfm_driver_rp.v.**

These procedures provide support for displaying messages in a common format, suppressing informational messages, and stopping simulation on specific message types.

The following constants define the type of message and their values determine whether a message is displayed or simulation is stopped after a specific message. Each displayed message has a specific prefix, based on the message type in the following table.

You can suppress the display of certain message types. The default values determining whether a message type is displayed are defined in the following table. To change the default message display, modify the display default value with a procedure call to `ebfm_log_set_suppressed_msg_mask`.

Certain message types also stop simulation after the message is displayed. The following table shows the default value determining whether a message type stops simulation. You can specify whether simulation stops for particular messages with the procedure `ebfm_log_set_stop_on_msg_mask`.

All of these log message constants type `integer`.

Table 14-12: Log Messages

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
EBFM_MSG_DEBUG	Specifies debug messages.	0	No	No	DEBUG:
EBFM_MSG_INFO	Specifies informational messages, such as configuration register values, starting and ending of tests.	1	Yes	No	INFO:
EBFM_MSG_WARNING	Specifies warning messages, such as tests being skipped due to the specific configuration.	2	Yes	No	WARNING:
EBFM_MSG_ERROR_INFO	Specifies additional information for an error. Use this message to display preliminary information before an error message that stops simulation.	3	Yes	No	ERROR:
EBFM_MSG_ERROR_CONTINUE	Specifies a recoverable error that allows simulation to continue. Use this error for data miscompares.	4	Yes	No	ERROR:
EBFM_MSG_ERROR_FATAL	Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible.	N/A	Yes Cannot suppress	Yes Cannot suppress	FATAL:

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
EBFM_MSG_ERROR_FATAL_TB_ERR	Used for BFM test driver or Root Port BFM fatal errors. Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible. Use this error message for errors that occur due to a problem in the BFM test driver module or the Root Port BFM, that are not caused by the Endpoint Application Layer being tested.	N/A	Y Cannot suppress	Y Cannot suppress	FATAL:

ebfm_display Verilog HDL Function

The ebfm_display procedure or function displays a message of the specified type to the simulation standard output and also the log file if ebfm_log_open is called.

A message can be suppressed, simulation can be stopped or both based on the default settings of the message type and the value of the bit mask when each of the procedures listed below is called. You can call one or both of these procedures based on what messages you want displayed and whether or not you want simulation to stop for specific messages.

- When eb fm_log_set_suppressed_msg_mask is called, the display of the message might be suppressed based on the value of the bit mask.
- When ebfm_log_set_stop_on_msg_mask is called, the simulation can be stopped after the message is displayed, based on the value of the bit mask.

Location	altpcietb_bfm_driver_rp.v	
Syntax	Verilog HDL: dummy_return:=ebfm_display(msg_type, message);	
Argument	msg_type	Message type for the message. Should be one of the constants defined in Table 18–36 on page 18–41.
	message	The message string is limited to a maximum of 100 characters. Also, because Verilog HDL does not allow variable length strings, this routine strips off leading characters of 8'h00 before displaying the message.
Return	always 0	Applies only to the Verilog HDL routine.

ebfm_log_stop_sim Verilog HDL Function

The `ebfm_log_stop_sim` procedure stops the simulation.

Location	altpcietb_bfm_driver_rp.v	
Syntax	Verilog VHDL: `return:=ebfm_log_stop_sim(success);`	
Argument	`success`	When set to a 1, this process stops the simulation with a message indicating successful completion. The message is prefixed with `SUCCESS:`. Otherwise, this process stops the simulation with a message indicating unsuccessful completion. The message is prefixed with `FAILURE:`.
Return	Always 0	This value applies only to the Verilog HDL function.

ebfm_log_set_suppressed_msg_mask #Verilog HDL Function

The `ebfm_log_set_suppressed_msg_mask` procedure controls which message types are suppressed.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`bfm_log_set_suppressed_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE: EBFM_MSG_DEBUG].` A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to be suppressed.

ebfm_log_set_stop_on_msg_mask Verilog HDL Function

The `ebfm_log_set_stop_on_msg_mask` procedure controls which message types stop simulation. This procedure alters the default behavior of the simulation when errors occur as described in the *BFM Log and Message Procedures*.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_log_set_stop_on_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE:EBFM_MSG_DEBUG].` A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to stop the simulation after the message is displayed.

Related Information

BFM Log and Message Procedures on page 14-34

ebfm_log_open Verilog HDL Function

The `ebfm_log_open` procedure opens a log file of the specified name. All displayed messages are called by `ebfm_display` and are written to this log file as simulator standard output.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_log_open (fn)`	
Argument	`fn`	This argument is type `string` and provides the file name of log file to be opened.

ebfm_log_close Verilog HDL Function

The `ebfm_log_close` procedure closes the log file opened by a previous call to `ebfm_log_open`.

Location	altpcietb_bfm_driver_rp.v
Syntax	`ebfm_log_close`
Argument	NONE

Verilog HDL Formatting Functions

The Verilog HDL Formatting procedures and functions are available in the **altpcietb_bfm_driver_rp.v**. The formatting functions are only used by Verilog HDL. All these functions take one argument of a specified length and return a vector of a specified length.

himage1

This function creates a one-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument	`vec`	Input data type `reg` with a `range` of 3:0.
Return range	`string`	Returns a 1-digit hexadecimal representation of the input argument. Return data is type `reg` with a `range` of 8:1

himage2

This function creates a two-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 7:0.
Return range	`string`	Returns a 2-digit hexadecimal presentation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 16:1

himage4

This function creates a four-digit hexadecimal string representation of the input argument can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 15:0.
Return range	Returns a four-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 32:1.	

himage8

This function creates an 8-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type reg with a range of 31:0.
Return range	`string`	Returns an 8-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 64:1.

himage16

This function creates a 16-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type reg with a range of 63:0.
Return range	`string`	Returns a 16-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 128:1.

dimage1

This function creates a one-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 1-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 8:1. Returns the letter *U* if the value cannot be represented.

dimage2

This function creates a two-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 2-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 16:1. Returns the letter *U* if the value cannot be represented.

dimage3

This function creates a three-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	string	Returns a 3-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 24:1. Returns the letter *U* if the value cannot be represented.

dimage4

This function creates a four-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 4-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 32:1. Returns the letter *U* if the value cannot be represented.

dimage5

This function creates a five-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.

Location	altpcietb_bfm_driver_rp.v	
Return range	string	Returns a 5-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type reg with a range of 40:1. Returns the letter *U* if the value cannot be represented.

dimage6

This function creates a six-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to ebfm_display.

Location	altpcietb_bfm_log.v	
Syntax	string:= dimage(vec)	
Argument range	vec	Input data type reg with a range of 31:0.
Return range	string	Returns a 6-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type reg with a range of 48:1. Returns the letter *U* if the value cannot be represented.

dimage7

This function creates a seven-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to ebfm_display.

Location	altpcietb_bfm_log.v	
Syntax	string:= dimage(vec)	
Argument range	vec	Input data type reg with a range of 31:0.
Return range	string	Returns a 7-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type reg with a range of 56:1. Returns the letter <*U*> if the value cannot be represented.

Procedures and Functions Specific to the Chaining DMA Design Example

The procedures specific to the chaining DMA design example are in the Verilog HDL module file **altpcietb_bfm_driver_rp.v**.

chained_dma_test Procedure

The `chained_dma_test` procedure is the top-level procedure that runs the chaining DMA read and the chaining DMA write

Location	altpcietb_bfm_driver_rp.v	
Syntax	`chained_dma_test (bar_table, bar_num, direction, use_msi, use_eplast)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`direction`	When 0 the direction is read. When 1 the direction is write.
	`Use_msi`	When set, the Root Port uses native PCI Express MSI to detect the DMA completion.
	`Use_eplast`	When set, the Root Port uses BFM shared memory polling to detect the DMA completion.

dma_rd_test Procedure

Use the `dma_rd_test` procedure for DMA reads from the Endpoint memory to the BFM shared memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_rd_test (bar_table, bar_num, use_msi, use_eplast)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`Use_msi`	When set, the Root Port uses native PCI express MSI to detect the DMA completion.
	`Use_eplast`	When set, the Root Port uses BFM shared memory polling to detect the DMA completion.

dma_wr_test Procedure

Use the `dma_wr_test` procedure for DMA writes from the BFM shared memory to the Endpoint memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_wr_test (bar_table, bar_num, use_msi, use_eplast)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`Use_msi`	When set, the Root Port uses native PCI Express MSI to detect the DMA completion.
	`Use_eplast`	When set, the Root Port uses BFM shared memory polling to detect the DMA completion.

dma_set_rd_desc_data Procedure

Use the `dma_set_rd_desc_data` procedure to configure the BFM shared memory for the DMA read.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_set_rd_desc_data (bar_table, bar_num)`	
`Arguments`	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.

dma_set_wr_desc_data Procedure

Use the `dma_set_wr_desc_data` procedure to configure the BFM shared memory for the DMA write.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_set_wr_desc_data_header (bar_table, bar_num)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.

dma_set_header Procedure

Use the `dma_set_header` procedure to configure the DMA descriptor table for DMA read or DMA write.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_set_header (bar_table, bar_num, Descriptor_size, direction, Use_msi, Use_eplast, Bdt_msb, Bdt_lab, Msi_number, Msi_traffic_class, Multi_message_enable)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`Descriptor_size`	Number of descriptor.
	`direction`	When 0 the direction is read. When 1 the direction is write.
	`Use_msi`	When set, the Root Port uses native PCI Express MSI to detect the DMA completion.
	`Use_eplast`	When set, the Root Port uses BFM shared memory polling to detect the DMA completion.
	`Bdt_msb`	BFM shared memory upper address value.
	`Bdt_lsb`	BFM shared memory lower address value.
	`Msi_number`	When `use_msi` is set, specifies the number of the MSI which is set by the `dma_set_msi` procedure.
	`Msi_traffic_class`	When `use_msi` is set, specifies the MSI traffic class which is set by the `dma_set_msi` procedure.
	`Multi_message_enable`	When `use_msi` is set, specifies the MSI traffic class which is set by the `dma_set_msi` procedure.

rc_mempoll Procedure

Use the `rc_mempoll` procedure to poll a given dword in a given BFM shared memory location.

Location	altpcietb_bfm_driver_rp.v
Syntax	`rc_mempoll (rc_addr, rc_data, rc_mask)`

Location	altpcietb_bfm_driver_rp.v	
Arguments	rc_addr	Address of the BFM shared memory that is being polled.
	rc_data	Expected data value of the that is being polled.
	rc_mask	Mask that is logically ANDed with the shared memory data before it is compared with rc_data.

msi_poll Procedure

The msi_poll procedure tracks MSI completion from the Endpoint.

Location	altpcietb_bfm_driver_rp.v	
Syntax	msi_poll(max_number_of_msi,msi_address,msi_expected_dmawr,msi_expected_dmard,dma_write,dma_read)	
Arguments	max_number_of_msi	Specifies the number of MSI interrupts to wait for.
	msi_address	The shared memory location to which the MSI messages will be written.
	msi_expected_dmawr	When dma_write is set, this specifies the expected MSI data value for the write DMA interrupts which is set by the dma_set_msi procedure.
	msi_expected_dmard	When the dma_read is set, this specifies the expected MSI data value for the read DMA interrupts which is set by the dma_set_msi procedure.
	Dma_write	When set, poll for MSI from the DMA write module.
	Dma_read	When set, poll for MSI from the DMA read module.

dma_set_msi Procedure

The dma_set_msi procedure sets PCI Express native MSI for the DMA read or the DMA write.

Location	altpcietb_bfm_driver_rp.v
Syntax	dma_set_msi(bar_table, bar_num, bus_num, dev_num, fun_num, direction, msi_address, msi_data, msi_number, msi_traffic_class, multi_message_enable, msi_expected)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory.
	bar_num	BAR number to analyze.
	Bus_num	Set configuration bus number.
	dev_num	Set configuration device number.
	Fun_num	Set configuration function number.
	Direction	When 0 the direction is read. When 1 the direction is write.
	msi_address	Specifies the location in shared memory where the MSI message data will be stored.
	msi_data	The 16-bit message data that will be stored when an MSI message is sent. The lower bits of the message data will be modified with the message number as per the PCI specifications.
	Msi_number	Returns the MSI number to be used for these interrupts.
	Msi_traffic_class	Returns the MSI traffic class value.
	Multi_message_enable	Returns the MSI multi message enable status.
	msi_expected	Returns the expected MSI data value, which is msi_data modified by the msi_number chosen.

find_mem_bar Procedure

The find_mem_bar procedure locates a BAR which satisfies a given memory space requirement.

Location	altpcietb_bfm_driver_rp.v
Syntax	Find_mem_bar(bar_table,allowed_bars,min_log2_size, sel_bar)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory
	allowed_bars	One hot 6 bits BAR selection
	min_log2_size	Number of bit required for the specified address space
	sel_bar	BAR number to use

dma_set_rclast Procedure

The dma_set_rclast procedure starts the DMA operation by writing to the Endpoint DMA register the value of the last descriptor to process (RCLast).

Location	altpcietb_bfm_driver_rp.v	
Syntax	Dma_set_rclast(bar_table, setup_bar, dt_direction, dt_rclast)	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory
	setup_bar	BAR number to use
	dt_direction	When 0 read, When 1 write
	dt_rclast	Last descriptor number

ebfm_display_verb Procedure

The ebfm_display_verb procedure calls the procedure ebfm_display when the global variable DISPLAY_ALL is set to 1.

Location	altpcietb_bfm_driver_chaining.v	
Syntax	ebfm_display_verb(msg_type, message)	
Arguments	msg_type	Message type for the message. Should be one of the constants defined in *BFM Log and Message Procedures*.
	message	The message string is limited to a maximum of 100 characters. Also, because Verilog HDL does not allow variable length strings, this routine strips off leading characters of 8'h00 before displaying the message.

Related Information

BFM Log and Message Procedures on page 14-34

Setting Up Simulation

Changing the simulation parameters reduces simulation time and provides greater visibility.

Changing Between Serial and PIPE Simulation

By default, the Altera testbench runs a serial simulation. You can change between serial and PIPE simulation by editing the top-level testbench file.

The `serial_sim_hwtcl` and `enable_pipe32_phyip_ser_driver_hwtcl` parameters control whether simulation is in serial mode or PIPE simulation mode. The parameters are defined in the top-level testbench, ***<working_dir>*/ep_g1x8_tb/ep_g1x8_tb/sim/ep_g1x8_tb.v**.

Table 14-13: Controlling Serial and PIPE Simulations

Data Rates	Parameter Settings	
	serial_sim_hwtcl	enable_pipe32_phyip_ser_driver_hwtcl
Serial simulation	1	0
PIPE simulation	0	1

Using the PIPE Interface for Gen1 and Gen2 Variants

Running the simulation in PIPE mode reduces simulation time and provides greater visibility.

Complete the following steps to simulate using the PIPE interface:

1. Change to your simulation directory, ***<work_dir>*/*<variant>*/testbench/*<variant>*_tb/simulation**
2. Open ***<variant>*_tb.v**.
3. Search for the string, `serial_sim_hwtcl`. Set the value of this parameter to 0 if it is 1.
4. Save ***<variant>*_tb.v**.

Viewing the Important PIPE Interface Signals

You can view the most important PIPE interface signals, `txdata`, `txdatak`, `rxdata`, and `rxdatak` at the following level of the design hierarchy: `altpcie_<device>_hip_pipen1b|twentynm_hssi_<gen>_<lanes>_pcie_hip`.

Disabling the Scrambler for Gen1 and Gen2 Simulations

The encoding scheme implemented by the scrambler applies a binary polynomial to the data stream to ensure enough data transitions between 0 and 1 to prevent clock drift. The data is decoded at the other end of the link by running the inverse polynomial.

Complete the following steps to disable the scrambler:

1. Open **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/altpcie_tbed_<*dev*>_hwtcl.v**.
2. Search for the string, `test_in`.
3. To disable the scrambler, set `test_in[2] = 1`.
4. Save **altpcie_tbed_sv_hwtcl.v**.

Disabling 8B/10B Encoding and Decoding for Gen1 and Gen2 Simulations

You can disable 8B/10B encoding and decoding to facilitate debugging.

For Gen1 and Gen2 variants, you can disable 8B/10B encoding and decoding by setting `test_in[2] = 1` in **altpcietb_bfm_top_rp.v**.

Debugging 15

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

As you bring up your PCI Express system, you may face a number of issues related to FPGA configuration, link training, BIOS enumeration, data transfer, and so on. This chapter suggests some strategies to resolve the common issues that occur during hardware bring-up.

Simulation Fails To Progress Beyond Polling.Active State

If your PIPE simulation cycles between the Detect.Quiet, Detect.Active, and Polling.Active LTSSM states, the PIPE interface width may be incorrect. The width DUT top-level PIPE interface is 32 bits for Arria 10 devices.

Make the changes shown in the following table for the 32-bit PIPE interface.

Table 15-1: Changes for 32-Bit PIPE Interface

8-Bit PIPE Interface	32-Bit PIPE Interface
`output wire [7:0] pcie_a10_hip_0_hip_pipe_txdata0`	`output wire [31:0] pcie_a10_hip_0_hip_pipe_txdata0`
`input wire [7:0] pcie_a10_hip_0_hip_pipe_rxdata0`	`input wire [31:0] pcie_a10_hip_0_hip_pipe_rxdata0`
`output wire pcie_a10_simulation_inst_pcie_a10_hip_0_hip_pipe_txdatak0`	`output wire [3:0] pcie_a10_simulation_inst_pcie_a10_hip_0_hip_pipe_txdatak0`
`input wire pcie_a10_simulation_inst_pcie_a10_hip_0_hip_pipe_rxdatak0`	`input wire [3:0] pcie_a10_simulation_inst_pcie_a10_hip_0_hip_pipe_rxdatak0`

Hardware Bring-Up Issues

Typically, PCI Express hardware bring-up involves the following steps:

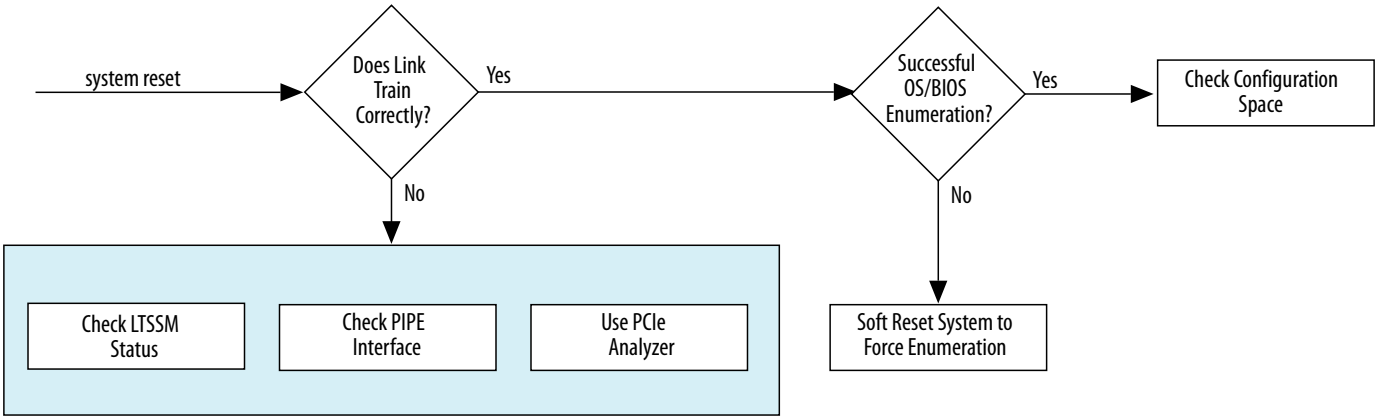
1. System reset
2. Link training
3. BIOS enumeration

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

The following sections, describe how to debug the hardware bring-up flow. Altera recommends a systematic approach to diagnosing bring-up issues as illustrated in the following figure.

Figure 15-1: Debugging Link Training Issues



Link Training

The Physical Layer automatically performs link training and initialization without software intervention. This is a well-defined process to configure and initialize the device's Physical Layer and link so that PCIe packets can be transmitted. If you encounter link training issues, viewing the actual data in hardware should help you determine the root cause. You can use the following tools to provide hardware visibility:

- SignalTap II Embedded Logic Analyzer
- Third-party PCIe analyzer

You can use SignalTap II Embedded Logic Analyzer to diagnose the LTSSM state transitions that are occurring on the PIPE interface. The `ltssmstate[4:0]` bus encodes the status of LTSSM. The LTSSM state machine reflects the Physical Layer's progress through the link training process. For a complete description of the states these signals encode, refer to *Status, Link Training and Reset Signals*. When link training completes successfully and the link is up, the LTSSM should remain stable in the L0 state. When link issues occur, you can monitor `ltssmstate[4:0]` to determine the cause.

Related Information

Reset

Use Third-Party PCIe Analyzer

A third-party logic analyzer for PCI Express records the traffic on the physical link and decodes traffic, saving you the trouble of translating the symbols yourself. A third-party logic analyzer can show the two-way traffic at different levels for different requirements. For high-level diagnostics, the analyzer shows the LTSSM flows for devices on both side of the link side-by-side. This display can help you see the link training handshake behavior and identify where the traffic gets stuck. A traffic analyzer can display the contents of packets so that you can verify the contents. For complete details, refer to the third-party documentation.

BIOS Enumeration Issues

Both FPGA programming (configuration) and the initialization of a PCIe link require time. Potentially, an Altera FPGA including a Hard IP block for PCI Express may not be ready when the OS/BIOS begins enumeration of the device tree. If the FPGA is not fully programmed when the OS/BIOS begins its enumeration, the OS does not include the Hard IP for PCI Express in its device map.

You can use either of the following two methods to eliminate this issue:

- You can perform a soft reset of the system to retain the FPGA programming while forcing the OS/BIOS to repeat its enumeration.
- You can use CvP to program the device.

Frequently Asked Questions

A

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

The following miscellaneous facts might be of assistance in troubleshooting:

- Only the Root Ports can be loopback masters.
- Refer to Altera Solution **rd04242015_385** for information about required QSF assignments you must add to your Gen3 project for Arria 10 ES2 devices in the 15.0 release.
- Refer to other Altera Solutions by searching in the Knowledge Base under Support on the Altera website.

Related Information

- **Known issues for Arria 10 PCIe solutions**
- **Required QSF assignments for Arria 10 ES2 Gen3 projects**
- **General Arria 10 PCIe Solution questions and answers**

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Lane Initialization and Reversal

B

2015.06.05

UG-01145_avmm

Subscribe Send Feedback

Connected components that include IP blocks for PCI Express need not support the same number of lanes. The ×4 variations support initialization and operation with components that have 1, 2, or 4 lanes. The ×8 variant supports initialization and operation with components that have 1, 2, 4, or 8 lanes.

Lane reversal permits the logical reversal of lane numbers for the ×1, ×2, ×4, and ×8 configurations. Lane reversal allows more flexibility in board layout, reducing the number of signals that must cross over each other when routing the PCB.

Table B-1: Lane Assignments without Lane Reversal

Lane Number	7	6	5	4	3	2	1	0
×8 IP core	7	6	5	4	3	2	1	0
×4 IP core	—	—	—	—	3	2	1	0
×1 IP core	—	—	—	—	—	—	—	0

Table B-2: Lane Assignments with Lane Reversal

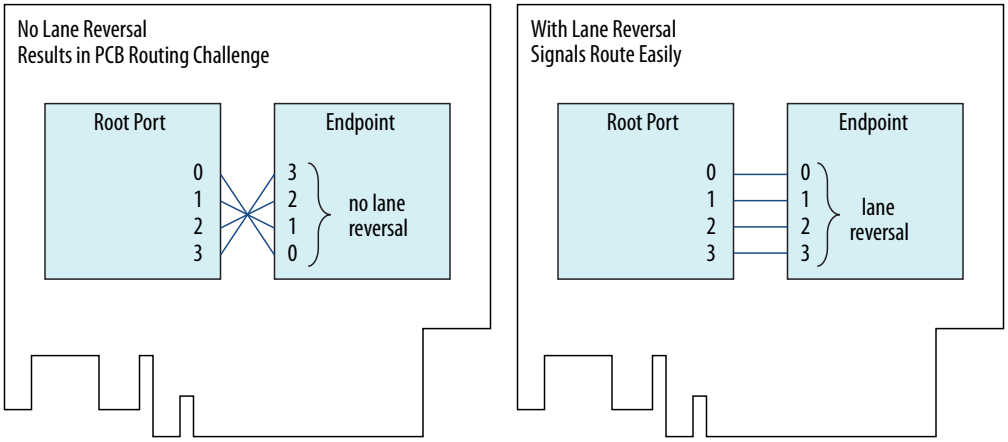
Core Config	8				4				1			
Slot Size	8	4	2	1	8	4	2	1	8	4	2	1
Lane pairings	7:0,6:1,5:2, 4:3, 3:4,2:5, 1:6,0:7	3:4,2:5, 1:6,0:7	1:6, 0:7	0:7	7:0,6:1, 5:2,4:3	3:0,2:1, 1:2,0:3	3:0, 2:1	3:0	7:0	3:0	1:0	0:0

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure B-1: Using Lane Reversal to Solve PCB Routing Problems

The following figure illustrates a PCI Express card with ×4 IP Root Port and a ×4 Endpoint on the top side of the PCB. Connecting the lanes without lane reversal creates routing problems. Using lane reversal solves the problem.



Additional Information

C

2015.06.05

UG-01145_avmm

Subscribe

Send Feedback

Revision History for the Avalon-MM Interface

Date	Version	Changes Made
2015.06.05	15.0	Added note in **Physical Layout of Hard IP In Arria 10 Devices** to explain Arria 10 design constraint that requires that if the lower HIP on one side of the device is configured with a Gen3 x4 or Gen3 x8 IP core, and the upper HIP on the same side of the device is also configured with a Gen3 IP core, then the upper HIP must be configured with a x4 or x8 IP core.
2015.05.14	15.0	Made the following changes to the user guide: • Added **Enable Hard IP Status Bus when using the AVMM interface** parameter in **Interface System Settings** on page 3-4. This parameter is available in the IP core v15.0 and later.
2015.05.04	15.0	Made the following changes to the user guide: • Enhanced the descriptions in **Avalon-MM-to-PCI Express Address Translation Table** on page 6-18. • Added **Enable Altera Debug Master Endpoint (ADME)** parameter to support optional Native PHY register programming with the Altera System Console. • Added support to send message TLPs with data payload of any length from a Root Port. Refer to **Programming Model for Avalon-MM Root Port** on page 6-26 and to the new supported TLP entry for Avalon-MM variations in the Feature Comparison for all Hard IP for PCI Express IP Cores table in **Features** . • Added information about the new custom example designs, in **Example Designs** on page 1-9. • Added column for Avalon-ST Interface with SR-IOV variations in Feature Comparison for all Hard IP for PCI Express IP Cores table in **Features** section.

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Date	Version	Changes Made
		• Enhanced descriptions of channel placement, added fPLL placement for Gen1 and Gen2 data rates, and added master CGB location, in **Physical Layout of Hard IP In Arria 10 Devices** on page 4-1. • Updated DUT module name in testbench and example design figures. • Removed list of static example designs from **Example Designs** on page 1-9. You can derive the list from the installation directory where example designs are available. • Removed Migration and TLP Format appendices, and added new appendix **Frequently Asked Questions** on page 16-1. • Reorganized sections in **Debugging** on page 15-1 and **Setting Up Simulation** on page 14-49. Removed *Reducing Counter Values for Serial Simulations* section, which is no longer relevant. Default counter values are automatically set for simulation. • Updated information in **SDC Timing Constraints** on page 11-2. • Fixed minor errors and typos.
2014.12.15	14.1	Made the following changes: • Revised Root Port programming model description, *Receiving a Completion TLP*, to cover read and non-posted completions. • Added *Avalon-MM Testbench and Design Example* chapter. • Added statement that the bottom left hard IP block includes the CvP functionality for flip chip packages. For other package types, the CvP functionality is in the bottom right block. • Corrected bit definitions for CvP Status register. • Updated definition of `CVP_NUMCLKS` in the CvP Mode Control register. • Added definitions for `test_in[2]`, `test_in[6]`, and `test_in[7]`. • Revised discussion of SDC files to include in Quartus II project.
2014.08.18	14.0 Arria 10	Made the following changes to the Arria 10 Avalon-MM Hard IP for PCI Express

Date	Version	Changes Made
		• Optionally changed the `cra_address` to 14 bits from 12. • Added simulation log file, **altpcie_monitor_a10_dlhip_tlp_file_log.log**, that is automatically generated in your simulation directory. To simulate in the Quartus II 14.0 software release, you must regenerate your IP core to create the supporting monitor file the generates **altpcie_monitor_a10_dlhip_tlp_file_log.log**. Refer to *Understanding Simulation Dump File Generation* for details. • Added support for 64-bit addressing, making address translation unnecessary. • Removed *Channel Placement for PCIe in Arria 10 Devices*. Please contact your Altera sales representative for PLL and channel usage. • Added simulation support for Phase 2 and Phase 3 equalization when requested by third-party BFM. • Added restrictions on the legal patterns of enabled and disabled bytes for `txs_byteenable[<w>-1:0]`. • Changed the PIPE interface to 32 bits for all data rates. This change requires you to recompile your 13.1 variant in 14.0. Made the following changes to the user guide: • Changed device part number for *Getting Started* chapter to **10AX115R2F40I2LG**. • Corrected frequency range for `hip_reconfig_clk`. It should be 100-125 MHz. • Clarified the behavior of the `txs_waitrequest` signal. • Added statement that the bottom left hard IP block includes the CvP functionality for flip chip packages. For other package types, the CvP functionality is in the bottom right block. • Simplified the *Getting Started* chapter. It copies the example from the install directory and does not include step-by-step instructions to recreate the design. • Removed 125 MHz clock as optional `refclk` frequency in Arria 10 devices. Arria 10 devices support an 100 MHz reference clock as specified by the *PCI Express Base Specification, Rev 3.0*. • Added definitions for `test_in[2]`, `test_in[6]` and `test_in[7]`. • Clarified that the Avalon-MM Bridge does not generate out-of-order Avalon-MM-to-PCI Express Read Completions even to different BARs. • Added sections on making analog QSF and pin assignments. • Enhanced the definition of Device ID and Sub-system Vendor ID to say that these registers are only valid in the Type 0 (Endpoint) Configuration Space. • Updated *Power Supply Voltage Requirements* table.

Date	Version	Changes Made
		• Removed all references to the Avalon-MM interrupt vector register. This register is not used. • Corrected values for **Maximum payload size** parameter. The sizes available are 128 or 256 bytes. • Removed `txdatavalid0` signal from the PIPE interface. This signal is not available. • Updated *Power Supply Voltage Requirements* table. • Updated *Physical Placement of the Arria 10 Hard IP for PCIe IP and Channels* to show GT devices instead of GX devices. • Corrected bit definitions for `CvP Status` register. • Updated definition of `CVP_NUMCLKS` in the `CvP Mode Control` register. • Removed discussion of `pclk`. This clock is not customer accessible in Arria 10 devices. • Removed PLL from channel placement figures. • Added fast passive parallel (FPP) to supported configuration schemes in *CvP in Arria 10 Devices* figure. • Corrected *Reset Controller in Arria 10 Devices* figure in *Reset and Clocks* chapter. • Corrected bit definitions for `CvP Status` register.
2013.12.02	13.1 Arria 10	Initial release.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

1. You can also contact your local Altera sales office or sales representative.

Related Information

- Technical Support
- Technical Training
- Customer Training
- Product Documentation
- Non-Technical Suport (general)
- Licensing

Typographic Conventions

The following table shows the typographic conventions this document uses.

Table C-1: Visual CueMeaning

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, \ **qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, <*file name*> and <*project name*> **.pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."

Visual Cue	Meaning
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
r	An angled arrow instructs you to press the Enter key.
1., 2., 3., anda., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
	Bullets indicate a list of items when the sequence of the items is not important.
1	The hand points to information that requires special attention.
h	The question mark directs you to a software help system with related information.
f	The feet direct you to another document or website with related information.
m	The multimedia icon directs you to a related multimedia presentation.
c	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
w	A warning calls attention to a condition or possible situation that can cause you injury.
The **Subscribe** button links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.	

Visual Cue	Meaning
	The **Feedback** icon allows you to submit feedback to Altera about the document. Methods for collecting feedback vary as appropriate for each document.

Related Information

Email Subscription Management Center